

Time and Resource Critical Virtualization

For the RIPE NCC box

LUIGI CORSELLO

RIPE NCC, HOGESCHOOL VAN AMSTERDAM

JANUARY 2013

ABSTRACT

The GII department and the Science Division at RIPE NCC would like to know if Atlas Anchor Nodes part of RIPE ATLAS could be hosted together with other Linux services in the same hardware system, provisionally named RIPE NCC box, by implementing *virtualization technology*.

Such technology should comply with at least three important requirements:

- **Accurate Timekeeping** with a precision relevant to the ATLAS anchor project.
- System and Network **Resource Separation and Availability** in case of load.
- **Scalable Manageability**: for GII to be able to remotely maintain multiple nodes.

Three virtualization candidates have been considered:

- **VMware vSphere Hypervisor 5.1 (ESXi)**: a commercial product of VMware Inc, freely usable in certain cases; offers full virtualization technology, is already used at RIPE NCC.
- **Open Virtuozzo (OpenVZ)**: open source, backed by Parallels, Inc; OS based virtualization similar to FreeBSD jails already in use at the GII department.
- **KVM**: open source supported by RedHat Inc; integrated in Linux and fully customizable; offers full virtualization with paravirtualization.

Extensive tests and benchmarks have been run, in a five months period between 2012 and 2013:

- **System benchmarks** through the *Phoronix Test Suite* and *Unixbench*.
- **Network tests** using the packet generation tool *iperf*.
- **Combined Network and System tests**, using *iperf* and the *stress* load generator.

The research brought to the following conclusions:

- The **ESXi** Hypervisor offers *very good scalability* and *acceptable resource separation* also in the network. However, high memory load interferes with the network stack and with an otherwise *relatively stable timekeeping*: an issue that needs further investigation.
- The **OpenVZ** solution offers *good scalability*, *acceptable system resource separation* with mediocre Disk and Memory performance, and *acceptable time stability* but it *fails on network resource separation*. Using separate network interfaces may overcome that.
- **KVM** was not stable under a paravirtualized clock source, *fails the timekeeping* requirement during intensive Disk I/O, and also *fails network resource separation*. *Scalability* also scores *mediocre* with KVM, that in general seemed not mature enough.

Additionally, hosting production services on *virtualization host* of KVM or OpenVZ is not advisable.

There is *no clear winner or optimal solution* that meets all requirements of this research. Conditional to legal confirmation about its license, and required further investigation on the memory subsystem: best choice could be *VMware*. Second choice for a quick implementation is *OpenVZ*.

A side result of the research is that relying on NTP for critical timekeeping using shared network resources may fail the strict requirement of accuracy, under heavy load. With or without virtualization. A review of the tolerable margins of "accurate timekeeping" for the *RIPE NCC box* is recommended.

This document describes the project *Virtualization for the RIPE NCC box* and its outcome, including research activities and technical highlights. An extensive report attached contains detailed results.

TABLE OF CONTENTS

1. INTRODUCTION.....	4
1.1. MOTIVE.....	4
1.2. DOCUMENT BUILDUP.....	4
1.2.1. ASSOCIATED DOCUMENTS.....	5
1.3. RESEARCH ETHICS.....	5
1.4. ACKNOWLEDGEMENTS.....	6
2. VIRTUALIZATION FOR THE RIPE NCC BOX.....	7
2.1. SECTION INTRODUCTION.....	7
2.2. THE RIPE NCC.....	7
2.2.1. THE SCIENCE DIVISION.....	7
2.2.2. THE RIPE ATLAS PROJECT.....	9
2.2.3. RIPE ATLAS ANCHOR NODES.....	9
2.3. PROJECT SCOPE.....	10
2.3.1. TECHNICAL REQUIREMENTS.....	11
2.3.2. QUESTIONS TO ANSWER.....	11
2.3.3. LIMITATIONS.....	12
2.4. PROJECT PHASES.....	12
2.5. THE PREPARATORY PHASE.....	14
2.5.1. INITIAL INTERVIEWS.....	14
2.5.2. CANDIDATE SERVICES.....	14
2.5.3. PROJECT INITIATION DOCUMENT.....	15
2.6. THE THEORETICAL RESEARCH PHASE.....	16
2.6.1. VIRTUALIZATION CANDIDATES.....	16
2.6.2. MONITORING AND TEST METRICS.....	17
2.6.3. TEST TYPES.....	18
2.7. THE PRACTICAL RESEARCH PHASE.....	19
2.7.1. TEST INFRASTRUCTURE (THE POC LAB).....	19
2.7.2. TEST EXECUTION.....	20
3. THE TECHNICAL SIDE.....	22
3.1. SECTION INTRODUCTION.....	22
3.2. THE CHOSEN VIRTUALIZATION TECHNOLOGIES.....	22
3.2.1. VMWARE VSPHERE HYPERVISOR 5.1 (ESXi).....	22
3.2.2. OPENVZ (OPEN VIRTUOZZO).....	23
3.2.3. KVM (KERNEL BASED VIRTUALIZATION).....	24
3.2.4. THE EXCLUDED VIRTUALIZATION TECHNOLOGIES.....	26
3.3. SETTING UP THE POC LAB.....	27
3.3.1. POC LAB HARDWARE AND VIRTUALIZED GUESTS.....	27
3.3.2. NETWORK CONNECTIVITY.....	28
3.3.3. SYSTEM DEPLOYMENT AND THE DRAC CHALLENGE.....	29

3.3.4.	VMWARE VSPHERE 5.1 AND DELL R320 HARDWARE.....	30
3.3.5.	OPERATING SYSTEM SETUP.....	30
3.3.6.	NTP CONFIGURATION.....	31
3.3.7.	THE KVM-CLOCK PARAVIRTUALIZED CLOCKSOURCE.....	32
3.4.	SYSTEM AND NETWORK RESOURCES MONITORING.....	33
3.4.1.	MONITORING LIMITATIONS.....	34
3.5.	MEASURING TIMEKEEPING STABILITY.....	34
3.5.1.	TIMEKEEPING QUIRKS AND POLLING INTERVAL LIMITATION.....	36
3.5.2.	SHARED NETWORK LIMITATION AND USE OF SEPARATE INTERFACES.....	38
3.6.	MEASURING RESOURCE SEPARATION.....	39
3.6.1.	TESTING THE VIRTUALIZATION HOST.....	39
3.6.2.	SYSTEM BENCHMARKS AND LOAD GENERATION.....	40
3.6.3.	NETWORK TESTS.....	41
3.6.4.	COMBINED TESTS.....	43
3.7.	AUTOMATED TEST SCHEDULING, DATA FETCHING AND GRAPH GENERATION.....	43
4.	OUTCOME AND ADVICE.....	45
4.1.	SECTION SUMMARY.....	45
4.2.	RESEARCH OUTCOME.....	45
4.3.	ACCURATE TIMEKEEPING.....	47
4.3.1.	VMWARE VSPHERE HYPERVISOR (ESXi).....	47
4.3.2.	OPENVZ.....	48
4.3.3.	KVM.....	48
4.3.4.	THE CRITICALITY OF NTP TIMEKEEPING.....	48
4.4.	RESOURCE SEPARATION.....	50
4.4.1.	VMWARE VSPHERE HYPERVISOR (ESXi).....	50
4.4.2.	OPENVZ.....	51
4.4.3.	KVM.....	53
4.4.4.	THE INTERFERENCE BETWEEN MEMORY AND NETWORK IN HYPERVISORS.....	54
4.5.	SCALABLE MANAGEABILITY.....	55
4.5.1.	VMWARE VSPHERE HYPERVISOR.....	55
4.5.2.	OPENVZ.....	56
4.5.3.	KVM.....	56
4.5.4.	THE MIXED BLESSINGS OF A VIRTUALIZATION HOST.....	57
4.6.	ADDITIONAL REQUIREMENTS.....	58
4.6.1.	"SCALABLE UPDATES" (DIFFERENT KERNELS ON EACH GUEST).....	58
4.6.2.	COMPLIANCE WITH EXISTING RIPE NCC SERVICES.....	58
4.7.	OUTCOME AND FINAL ADVICE.....	59
4.7.1.	SHORT TERM ACTION.....	60
4.7.2.	LONG TERM PERSPECTIVE.....	61
4.8.	ADDITIONAL REMARKS AND SIDE NOTES.....	62
4.8.1.	CPU HYPER-THREADING AND TURBO BOOST FEATURES.....	62
4.8.2.	ABOUT THE VSPHERE HYPERVISOR LICENSE.....	63
4.8.3.	ABOUT THE HARDWARE SPECIFICATION.....	63
4.9.	OPEN QUESTIONS.....	64
4.10.	VIRTUALIZATION PROTOTYPE AND FUTURE OF THE POC LAB.....	65
5.	BIBLIOGRAPHY.....	66

1. INTRODUCTION

1.1. MOTIVE

This is the final report of a graduation project that I carried out for RIPE NCC in Amsterdam between September 2012 and January 2013: *Time and Resource Critical Virtualization for the RIPE NCC box*.

My name is Luigi Corsello, part time student at the *Hogeschool van Amsterdam* since 2009 and ICT professional since 1999. I have senior experience in System and Network Engineering and junior experience as Project Manager, played at a number of Italian, Dutch and American companies across a period of 11 years.

This project report will focus on three aspects: research activities and outcome; technical implementation details; planning and project management aiding the delivery in a relatively short timeframe.

Next to delivering the results of the research, my aim is to demonstrate my maturity as IT professional and graduating student to set up and manage a business process; employ all necessary technical and non-technical means, deliver a product that meets the requirements.

1.2. DOCUMENT BUILDUP

Beyond the introductory section, this paper is split into *four more sections* and is meant for different kinds of audience that can selectively choose to read the sections that more concern their interest:

- **Section 2** reports about the research: *what to deliver*; the *phases* of the project; what work was done and which choices were made.
- **Section 3** presents the *technical implementation details*: the infrastructure setup, technical solutions used and so forth.
- **Section 4** focuses on *the outcome* of the research, containing brief results highlights, conclusions, side notes and the advice given.

- **Section 5** (*HvA version only*) contains some project management details and some personal reflections about *the process* that brought the project to completion. How it came into existence and how it was managed, some lessons learned and some points to improve.

Each section tries to follow a chronological order from its own different perspective: from the first activities and technical highlights down to the final delivery.

Document and attachments are written in English as agreed with all parts involved. RIPE NCC requires all company related papers to be written in English.

1.2.1. ASSOCIATED DOCUMENTS

- **Attachment 1** – *Time and Resource critical Virtualization for the RIPE NCC box – Results-Attachments* publishing the results of tests and benchmarks run during this research. It contains result tables, comparative graphs, result graphs and more. *This document is available on request from the RIPE NCC.*
- **Attachment 2** (*HvA version only*) – *Virtualization for the RIPE NCC box: Project Initiation Documentation (PID)*: contains project management related items: a *Business Case* and a *Plan of Approach* for the project inclusive of planning, activities, risk evaluation.

1.3. RESEARCH ETHICS

I have executed this research autonomously, while keeping my customer informed on a regular basis. Excluding remarks and requests from the customer, quality control was also in my hands.

Lacking external quality control on the very many items treated, no guarantee can be given that no errors have been made. However, the maximum care has been taken to prevent them or eventually correct them to my best effort.

Research ethics and absolute openness have been a central point throughout the project. All tests and benchmarks were executed taking care that they could not be affected by external interferences or technical glitches. When that happened, like by the first attempts of running automated tests, entire test sets have been repeated as many times as necessary until a consistent and reliable result was obtained. Exceptions and uncontrolled changes have been avoided for as much as possible.

The care for quality came to surface for example when testing in advance the effective functionality of system benchmarks, or double checking dubious results through multiple sources of information, or by making sure to always have two monitoring resources for the most important metrics, for confirmation.

Raw data results, however at the moment may not be in a format that can be easily understood by the general public, can be made available for control and verification. On request they can be published in a clearer way or orderly packaged and stored.

1.4. ACKNOWLEDGEMENTS

This document is dedicated to my father and my family in Italy who supported me for many years to my graduation despite all odds, even from the distance when I decided to move to the Netherlands.

A mention goes the personnel of the *Media, Creation and Information* department of the *Hogeschool van Amsterdam* devoted to part time courses and to my tutors, for their dedication and for offering me that bit of extra faith in succeeding, when needed.

Many thanks to the RIPE NCC with its unmistakable family feeling and research oriented attitude, for trusting my skills and offering me the chance to complete my studies. Special thanks for the sincere availability of all colleagues who supported my work.

2. VIRTUALIZATION FOR THE RIPE NCC BOX

2.1. SECTION INTRODUCTION

This section describes the activities of the project *Time and Resource Critical Virtualization for the RIPE NCC box* sponsored by the *GII department* at the *Science Division* of RIPE NCC.

The section begins with a short background on the company and department where the project took place, with an overview of their services involved in the project. Scope and requirements are then explained. A sketch of the planning and then all project phases are described. The most important research phase, split into a theoretical and a practical step fills the rest of the section.

2.2. THE RIPE NCC

The RIPE NCC, as its website currently states: "*is one of five Regional Internet Registries (RIRs) providing Internet resource allocations, registration services and coordination activities that support the operation of the Internet globally.*" Among its core services there are registration and assignment of Internet resources, IP addresses and AS numbers, plus a number of services for its members. Functional to its activities is the RIPE database holding all registration information for Internet resources in its service region. The RIPE NCC also manages the infrastructure of K-ROOT servers, one of the sets of servers that are on top of the Domain Name Resolution (DNS) tree, essential for the functioning of the Internet as we know it.

The NCC is a non-profit, neutral organization that was officially born in 1992 as the operative arm of RIPE (Réseaux IP Européens) a body established in 1989 to promote cooperation between network operators in the European region. The RIPE has also set up a long established decisional process where its community can influence the operational policies made. The *RIPE Meetings*, held at different locations throughout the year, always attract much attention from operators and policymakers.

2.2.1. THE SCIENCE DIVISION

The RIPE NCC historically had a research-oriented mentality, an attitude that remained intact across the years. In 2012, the *Science Division* department is busy with research, development and administration of a set of services for its member's community and for the Internet user base.

Internet Measurements have been an important area of interest of the department but not the only one. Here follows a shortlist of services of the Science Division possibly relevant to this project:

- *RIPEstat*, a web tool to collect and report information about different Internet resources by querying several data sources managed by the RIPE NCC (among which the RIPE Database).
- The *K-ROOT DNS servers infrastructure* already mentioned. The infrastructure is made of several anycast servers spread over different locations around the world. The use of anycast addressing makes it possible that every user attempting to contact a K-ROOT DNS server is always automatically routed to the nearest node.
- *DNSMON*, a service used to monitor the status of the Top Level Domain DNS servers of some countries (ccTLD) participating at the project and also the K-ROOT DNS servers, using software probes installed at different Internet locations. The probes are now hosted on Test Traffic Measurement (TTM) nodes but are being integrated into the RIPE ATLAS project (see below).
- *RIS*, or *Routing Information Service*, a route collector service well established for more than a decade, collecting Internet routing information through the BGP protocol from its nodes hosted at Internet Exchanges around the world. The information collected is stored and elaborated to provide statistics about IP address prefixes, AS numbers and their status on the global routing. Most information provided by RIS is reported through RIPEstat as well.
- *TTM*, or *Test Traffic Measurements* project, another long-standing service that can measure delay, packet loss, delay variation (jitter) and bandwidth between TTM nodes hosted by participating members. For precise bidirectional measurements the TTM nodes, based on the FreeBSD OS, are connected to a GPS antenna: GPS satellites provide excellent and consistent time accuracy across all nodes. They are also accessible for time synchronization as high precision "*stratum 1*" NTP servers. The TTM infrastructure is in the process of being phased out in favor of the more recent RIPE ATLAS project.

Most of the services mentioned are offered to RIPE NCC members (and to the general public as well in some cases) through the main website www.ripe.net. The Science Division also maintains the website labs.ripe.net for reports, findings and new ideas.

Head of the Science Division is *Daniel Karrenberg*, one of the minds behind RIPE who has had a role in the introduction of the Internet in Europe and is deeply involved with the Internet community.

The Science Division at RIPE NCC has two technical departments: *Global Information Infrastructure (GII)* is responsible for system and network engineering activities and for 24/7 availability of both remote nodes and local infrastructure backend, hosted in datacenters; *Research and Development (R&D)* improves existing services and develops new ones; A third department, "*Measurements Community Building*", is busy with project management and in the liaison between the operative departments and the community members.

The *GII department*, sponsor of this project, tightly cooperates with the *Operations Departments* external to the Science Division and responsible for the IT infrastructure of the entire RIPE NCC, including common services, physical and virtual servers and datacenter management.

2.2.2. THE RIPE ATLAS PROJECT

The RIPE ATLAS project, further referred to as *Atlas*, is the youngest and at the time of writing the fastest growing of the services of the Science Division. Its aim is to build a large network of internet measurements done by tiny embedded hardware probes, easily deployable at any location with internet connectivity and an USB power source.

Hardware probes autonomously schedule and perform a series of standard measurements using ICMP and UDP based tools (*ping*, *traceroute*); they can also be setup to perform extended measurements like using HTTP and DNS protocols. The K-ROOT server infrastructure is among the measurements targets of the *Atlas probes*.



Image 1: RIPE ATLAS Probe v.2

The information produced by *Atlas probes* is collected by RIPE NCC and analyzed to produce certain statistics, for example to observe how natural or political events affect the Internet infrastructure or to keep an eye on the "health of the Internet". The resulting measurements are publicly available. Probe hosts also have limited possibilities to configure their own probe(s) to perform specific tests, for example towards all probes of a specific group.

As of 2013, the *Atlas* project is moving out of its prototype phase. It can already count on more than 2000 active probes for now mostly in the European region and in other western countries while an equivalent number of probes are being distributed around the globe.

2.2.3. RIPE ATLAS ANCHOR NODES

Since Quarter 3, 2012, the Science Division has introduced the idea of *Atlas Anchor Nodes*, which brings us closer to the scope of this project. The current infrastructure of *Atlas probes* can produce statistics at a mostly global level: the next step would be having regional measurement targets to observe traffic patterns at a more fine-grained local level.

To reach that target *Anchor Nodes*, hosted at relevant Internet locations like at the network edge of *service providers*, could work both as big-size *software probes* and *measurement targets*, linking (or *anchoring*) a set of probes to a specific region. A software probe is a software package that can be installed on a system to perform the same functions as the hardware probes.

In addition to the standard *Atlas* measurements managed by the Science Division, RIPE NCC members who *host* an anchor, could then program their probes to serve specific purposes, like

measuring the performance of their own networks centered around the anchor node, or sending alerts in case something is wrong (for example in case certain targets become unreachable).

As of December 2012, the GII and R&D departments are running a pilot phase involving a limited number of participants hosting the first *Atlas Anchor nodes*.

GII engineers drafted standard *hardware specifications* for an *Anchor Node* for all participating hosts to order, rack mount and connect to their networks. After that, GII can remotely take control of the nodes and install operating system and all services needed on them, to manage them remotely.

The operating system used into *Atlas Anchor Nodes* (and at RIPE NCC in general) is the *CentOS Linux* distribution in its latest version, currently 6.3.

The pilot phase is proving useful for technical matters related to RIPE ATLAS measurements, like testing functionality and resource needs of the software probe, using an *Anchor Node* as a measurements target, inserting the *Anchor Nodes* into the Atlas data processing backend.

The pilot is also helping to settle political and administrative matters like the definitive hardware specifications to propose, handling support for the nodes, the process of registration of Anchor hosts and so forth.

The current plan is to gradually extend the number of hosts taking part to the pilot starting in *Quarter 1, 2013*, then gradually add all desired functionality and features needed until *Anchor Nodes* can be made fully operative and part of RIPE ATLAS public infrastructure.

2.3. PROJECT SCOPE

As we have seen, a number of different services offered by the RIPE NCC are based on remotely hosted server nodes, examples being *Atlas Anchor nodes* or *K-ROOT DNS servers*.

To consolidate certain services, single hardware nodes could host multiple services running independently from each other: that could be achieved by using virtualization technology.

The initial hardware specification for *Anchor nodes* was made with this idea in mind: a relatively powerful and standardized piece of hardware hosted at remote locations to serve as *Atlas Anchor node* but also host other present and future services. Such a node could be named **RIPE NCC box**.

Each service has its own specific requirements, for example: *Anchor nodes* require time accuracy to perform reliable measurements and each *K-ROOT* server instance should always count on CPU processing power and network bandwidth available, because potentially serving a large user base.

I have been asked by the manager of the GII department, *Romeo Zwart*, to research a number of virtualization technologies and verify if any of them could consistently comply with certain specific requirements in order to run certain services of RIPE NCC.

The research would take the *Atlas Anchor* project as a starting point and the hardware specification formulated for the *Atlas Anchor pilot phase* as reference hardware specification to use for testing.

2.3.1. TECHNICAL REQUIREMENTS

The primary technical requirements, already set at the beginning of this project and then carefully agreed upon in a preparatory phase of the project are specified in a *Project Initiation Document*. Here follows an excerpt:

- **Accurate time:** *the solution implemented must offer reliable time management: accurate timekeeping in the order of milliseconds is required; a more precise order of magnitude can be defined during the research.*
- **Resource separation and availability:** *service Containers or Virtual Machines should never interfere with each other in particular in case of CPU, network and disk I/O load and in terms of security. Allocated system resources must remain available to the Virtual Machines at any time.*
- **Scalable manageability:** *the RIPE NCC box and its services must be deployable and maintainable in a scalable, possibly semi-automated way.*
- **(Optional) scalable updates:** *it should be possible to perform system updates (like kernel updates) limited to one service platform without affecting the other services on a node.*
- **(Optional) compatibility:** *a node should support existing services without the need of a code overhaul. Exceptions like building pre-packaged versions of those services, for the sake of deployment on a node, are possible.*

2.3.2. QUESTIONS TO ANSWER

During a preparatory phase of the project, the requirements and primary scope of this project have been discussed with a number of interested parties in a series of *initial interviews*. Those have led to one main question this project attempts to answer. From the *Project Initiation Document*:

"Can a form of virtualization be implemented on a network node to reliably host certain Linux based services that require accurate time, resource separation and scalable manageability?"

The main question has led to a set of secondary questions to be answered. These formed the basis for the continuation of this project and the research associated to it:

- *Who shall be involved in the development of services for the nodes and in deployment and management of the nodes themselves?*
- *Which (model of) network services could be hosted on a node?*
- *What are technical and scalability requirements a node must comply with?*
- ***Which virtualization technologies could meet the requirements?***
- ***How to tune a virtualization technology to meet the requirements?***
- *What could be non-technical requirements for a node, and how to comply with them?*

The first three questions, already discussed at the beginning of the project, were answered in the preparatory phase (see next paragraphs). The two questions highlighted are the core questions of the project. Non technical requirements have to do with hardware and licensing requirements.

2.3.3. LIMITATIONS

As important as the main scope of the project were its limitations or *constraints*, also fixed into the *Project Initiation Document*. Here follows a summary:

- *No service implementation*: this project would test virtualization technologies against the requirements, not implement RIPE NCC services into virtual machines.
- *Compliance to current practices*: this project would attempt to comply and tune the research and its findings to the current practices at the GII department.
- *No policy handling*: policies and agreements between the RIPE NCC and its members for the implementation of virtualization on *Atlas Anchor Nodes* were out of scope.
- *No remote deployment, no implementation*: this project would not implement the advised virtualization technology beyond its own test infrastructure.

The limitation also set the main scope of this project to remain unchanged during its entire duration, also in case the plans of the *Science Division* would change.

2.4. PROJECT PHASES

This project has seen a *Preparatory phase* and a *Research Phase*. Those were freely based on the *Ten Steps Plan* for an advising research defined in ref [16] in the Bibliography. The *preparatory phase* ended when the *PID* was completed; this document concludes the *research phase*.

Table 1 shows a short summary of the main phases, their respective activities and their planned and effective duration. Despite the efforts *theoretical and practical research had to happen mostly in parallel*; also, *the practical phase required more activities than originally planned* with an additional stage of testing to be executed, causing some delay in the final delivery.

ACTIVITY	BEGIN	END	EFFECTIVE END
<i>Preparatory Phase</i>	3/Sep/2012	10/Oct/2012	
<i>Initial Interviews</i>	3/Sep/2012	18/Sep/2012	
<i>Analysis and Aggregation</i>	10/Sep/2012	18/Sep/2012	
<i>Activities Planning (PID)</i>	19/Sep/2012	28/Sep/2012	
<i>Agreement on Planning</i>	1/Oct/2012	10/Oct/2012	
<i>Research Phase</i>	1/Oct/2012	31/12/2012	
<i>Theoretical Research</i>	1/Oct/2012	29/Oct/2012	12/Dec/2012
<i>Practical Research</i>	11/Oct/2012	10/Dec/2012	21/Dec/2012
<i>Conclusions and Advice</i>	10/Dec/2012	31/12/2012	(est.) 15/1/2012

Table 1: Schematic project planning, with the effective end dates in the last column.

2.5. THE PREPARATORY PHASE

Building up the background of the project, acquiring the necessary knowledge about RIPE NCC and all preparations and agreements for the research phase happened during *the preparatory phase*.

2.5.1. INITIAL INTERVIEWS

This project was initially set up with two initial interviews with the manager GII, held before September 2012, to agree about the main lines explained earlier in this document.

The other initial interviews I held with:

- *Philip Homburg*, Atlas probe programmer at the R&D department, who contributed with ideas for time monitoring and provided the main requirements for both an Atlas probe and a measurement target for RIPE ATLAS.
- *Sean McAvoy*, and *John Bond*, senior engineers at the GII department, as interested parties for Atlas Anchor Nodes, who provided information about infrastructure and policies at GII and about the existing services of the Science Division.

Informal talks took place with *Anthony Anthony*, Atlas probe programmer, and *Anand Buddhdev*, senior GII engineer involved with the K-ROOT DNS server infrastructure. Of course more informal information exchange took place occasionally throughout the entire project.

Tight relations were kept with the engineers of the RIPE NCC *OPS (Operations) department*, especially for preparing and later setting up the POC infrastructure, to respect the internal policies of the company. The OPS department also provided information concerning the VMware infrastructure that they manage, under which a number of RIPE NCC backend resources are hosted.

2.5.2. CANDIDATE SERVICES

The initial interviews made clear which services could be hosted on the *RIPE NCC box*. As by the original agreements with the GII manager, the RIPE NCC box could only host *Linux*-based services of the *Science Division*, thus excluding the RIS project at this stage.

The most important initial candidate services came out to be:

- *An Atlas Anchor node*, initially functioning as software probe and anchor target, with potentially more functionalities in the future; R&D programmers remained responsible for developing software for an *Anchor Nodes* and add them to the RIPE ATLAS backend while *GII engineers* would take care of OS setup and maintenance of the nodes.

- An instance of K-ROOT DNS server, entirely managed by GII engineers.
- A *DNSMON probe*, soon to be integrated into the RIPE ATLAS framework by the R&D department.

Basic technical requirements for the services to host are worth mentioning in this section: each service would need at least the latest *CentOS Linux* distribution (that would be version 6.3) *two processor cores, two or four Gbytes RAM*. One limitation for an Atlas Software Probe to work was to have *no more than one network interface* active at a time, however that limitation might have been overcome by newer releases of the software probe package.

2.5.3. PROJECT INITIATION DOCUMENT

A big amount of information was collected during the initial interviews. That information was sorted and analyzed to form the basis of activities and planning in the Project Initiation Document. That document was further discussed and agreed upon with the manager GII and the supervisor for the Hogeschool van Amsterdam on October 10 2012, after which the core research activities of this project could be kicked off.

2.6. THE THEORETICAL RESEARCH PHASE

In the original planning the research part of this project was to be split into a *theoretical* and a *practical* phase. The practical phase would merely serve to perform tests and benchmarks planned during the theoretical phase. In reality, the two phases did happen in parallel, with the technical infrastructure for test execution and data collection being perfected as a consequence of the practical activities.

For the sake of clarity, the two research phases will be described separately. This paragraph will sketch an overview of the plans for tests and benchmarks.

The theoretical research had two scopes:

- Determine the virtualization technologies candidates to test.
- Design a test infrastructure and some methods to check the compliance of the virtualization candidates to the given requirements.

2.6.1. VIRTUALIZATION CANDIDATES

Choosing the virtualization candidates to test was a difficult challenge. It formed a *research into a research* for which many online resources were consulted. For the final choice, influenced by one of the technical requirements (*manageability*), the initial interviews were taken into consideration together with certain specific factors:

- *Affinity* with the RIPE NCC: does RIPE NCC already uses or has used a certain technology? Is there in-house experience and into which departments?
- *Compliance* with the current GII infrastructure: Can the GII department directly implement a certain technology or does that require relevant changes?
- *Cost*: is a certain technology free to use for the RIPE NCC box? Are there technical or license restrictions? Are there peculiar implementation challenges?
- *Differentiation*: one last scope was to test technically different technologies, to increase the chance to find a match with the requirements.

Three candidates came out of the choice: those are summarized in *Table 2*; the next section has more details about the chosen candidates. For proper reporting of the results, virtualization candidates are listed in the order they were implemented in the test lab, i.e. their order in *Table 2* and further in this document had no specific significance for the outcome of the research.

Table 2: Virtualization Candidates

VIRTUALIZATION	TECHNOLOGY	LICENSING	HIGHLIGHTS
VMware vSphere Hypervisor 5.1 (ESXi) Special build for Powerededge R320 provided by Dell Support	Full (Hypervisor) Virtualization	Commercial; free download; limited free usability.	<i>Already in use at the OPS department</i>
			<i>At least one senior engineer at GII has experience with it</i>
			<i>Full virtualization, a Virtual Machine offers similar facilities as a physical system</i>
OpenVZ Using vzkernel x86_64 version: 2.6.32-042stab065.3	Container (OS level) Virtualization	GPL v2	<i>Similar to "FreeBSD jails" already in use at GII; further no in-house experience with OpenVZ</i>
			<i>Near native access to system resources, in particular to a time source</i>
KVM On CentOS 6.3 Linux Kernel x86_64 version: 2.6.32-279.11.1 CentOS 6.3 qemu-kvm version: 0.12.1.2-2.295	Full Virtualization, "virtio" paravirtualized disk and network drivers	Differs per component: GPL, GPL v2, LGPL, LGPL v2.	<i>Highly tunable, favorite by system engineers. Limited in-house experience.</i>
			<i>Deeply integrated into the standard Linux Kernel.</i>
			<i>Paravirtualized elements should offer advantages in term of access to system resources for timekeeping and performance.</i>

2.6.2. MONITORING AND TEST METRICS

The research had two types of metrics: *network tests* were mostly evaluated through *empirical* observation of *monitoring* graphs of specific resources.

In addition to the monitoring graphs *system benchmarks* produced *concrete values and indexed scores* to better weight the differences. System benchmarks results could also allow the comparison between different technologies in some cases: an extra result offered by the research.

The monitoring was carefully set up on a separate host so that for each of the most important metrics there would be at least 2 monitoring tools collecting the same statistics, to always have a control value in case of uncertainty.

An important role in the research was played by the concept of **baselines**. Baselines are *the zero value result* or graphs for a specific metric drawn during a period of *idle or no load*, on each of the

systems tested. Later comparison of the baselines with the results under specific load could give a figure of the impact of that load compared to when the system was idle.

For example, some fluctuations in the accuracy of time caused by certain tests were minimal (in the order of hundreds of microseconds) but could be observed by comparing to the time baseline. Producing a timekeeping baseline of the physical virtualization hosts could help to mark the difference in the timekeeping between that physical host and the virtualized guests running on it.

The concept of baselines could also be applied to *benchmark results*: for example by benchmarking first a standalone guest (producing a "baseline") and then two guests simultaneously, to evaluate the difference in performance and in particular resource separation.

Especially important for this research were the metrics for **time**, **network** latency and **system resources** (CPU, etc). described in more detail in **Section 3** of this document.

2.6.3. TEST TYPES

One of the scopes of this project was to check resource separation.

Appropriate tools where necessary to first *generate simultaneous load in a predictable and repeatable way* but also *produce measurable results*, to assess how the performance of the systems was affected. That concept came out to be especially critical when testing *network resources*.

The tests executed used a combination of *freely available tools: benchmarking packages and network and system load generators* specifically engineered for this research. Perfecting and improving the automatic execution of tests and benchmarks required concrete efforts.

Three types of tests were planned and executed on both virtualization guests and Linux hosts:

- **System benchmarks** for CPU, DISK and Memory, would measure resource separation under simultaneous load but also allow performance comparison in certain cases.
- **Network tests** would produce UDP and TCP network flows and then observe their impact in many different use cases, through monitoring graphs.
- **Combined tests**, involving both System and Network for load generation, would allow to observe the interaction between network and system load and confirm previous findings.

The execution of tests has known **two stages**, the second stage only involving two remaining virtualization technologies that had passed the first stage. During the second stage, a reduced set of tests was executed in a much improved and better-automated setup.

Section 3 has a better description of tests and benchmarks implemented. All the details about test and benchmarking use cases are described in the *Results-Attachments* document.

2.7. THE PRACTICAL RESEARCH PHASE

The practical research phase can be roughly identified with the period during which tests and benchmarks were run and their results were collected, after the test infrastructure was made available.

2.7.1. TEST INFRASTRUCTURE (THE POC LAB)

The design of the test infrastructure or *POC Lab* started very early in the project to make sure the resources needed could be allocated in time for testing and benchmarking. The POC Lab was installed in a local workshop at the premises of RIPE NCC.

Like other items of this research, the test infrastructure was designed with number of specific factors in mind. All those were considered in the practical implementation:

- *Parallelism*: due to the limited time available, the POC Lab should allow the execution of as many tests and benchmarks as possible *in parallel* on all three virtualization candidates.
- *Consistency*: for the test results to be acceptable, the setup should remain consistent across all virtualization candidates implemented in the POC Lab.
 - All virtualized guests should share for as much as possible the same virtual resource assignment, operating system packages, system and services configurations and so on. Also to easily perform batch operations remotely on all of them.
 - For virtualization candidates running on top of a Linux host (*OpenVZ* and *KVM*): all hosts should also share the same system configuration and setup, like the guests did.
 - The hardware setup (HW specifications, BIOS setup, cabling, etc) should be consistently equal (this rule had one relevant exception, see next section).
- The entire POC Lab should retain the same setup until all tests and benchmarks of a stage were executed. This was meant to avoid that the reinstall of a system, the replacement of a network cable or a change in the monitoring software could taint an entire set of results.
- *Default Settings* as a rule. To respect the *scalability requirement*, all virtualization candidates were implemented for as much as possible using *their standard, recommended, or default settings*, without extensive tweaking. Only necessary configuration changes that the GII could potentially apply on remote RIPE NCC boxes on a large scale were allowed and recorded.
- *Realism*. The most important factor: the POC Lab should produce a situation for as much as possible similar to the realistic setup of a *RIPE NCC box* deployed at remote locations. In other words: the systems running on the virtualization guests should be similar to a standard system installed by GII through their standard practices. Also the resources assigned to virtualized guests should reflect the basic requirements for the candidate services (paragraph 2.5.2).

The POC Lab, completed before the first system tests of 19 November 2012, consisted of *three servers* running *one virtualization candidate each*. Every host had *two identical virtual guests*.

One central node was used to monitor and gather statistics for the entire infrastructure and as a *controller* to start automated test on schedule. Another node, installed like a *Test Traffic Measurement box*, provided a precise NTP time reference based on GPS, and was only meant to be used for time monitoring.

More details about the POC Lab are available in **Section 3**.

2.7.2. TEST EXECUTION

The execution of tests and benchmarks has been performed in two stages, the second stage added when more data was needed to draw clearer conclusions.

- In *stage 1*, all virtualization candidates were tested for basic compliance to the technical requirements. In case of system benchmarks, the extent of the tests in this stage did not produce comparable results among virtualization candidates but just *within them*, as needed to check the *resource separation* requirement. At the end of this stage one candidate was excluded from the research for clear lack of compliance to at least one requirement.
- During *stage 2*, run in December 2012, certain tests and benchmarks were re-executed after controlled changes to monitoring and resource assignment. Benchmark results produced during this stage also allowed performance comparison *among* the remaining virtualization candidates.

The status of testing activities has been shared within the *Science Division* at RIPE NCC. An extended webpage with all the scheduling has been published and kept up-to-date until all tests were executed. Some input from R&D developers and GII engineers helped to pinpoint better test practices and "most asked" results and led to the running of the second stage of tests.

Table 3 shows a short overview of the schedule of tests and benchmarks. Result tables of system benchmarks and comparative and monitoring graphs also for network and combined tests are shown in the **Results-Attachments** document associated to this report.

System benchmarks have mostly been executed in parallel on all candidates. Network and combined tests, on the contrary, had to be executed sequentially on one virtualization candidate at a time: the generation of network packet flows always required a client side and a server side and the hardware resources were insufficient to run multiple tests in parallel.

To facilitate the execution of several rounds of consecutive tests, running network and combined tests was fully automated through shell scripts: the automated scheduling was controlled by a central server.

A log of the tests executed on schedule was saved in a text file. That file was later parsed to automatically fetch or generate all relevant graphs for each specific test in the specific timeframe when it was run and for the specific hosts on which it was run.

Thanks to the automation, all graphs for each test executed could be saved into a generated tree of directories that was published on an internal web server. This automation made available thousands of graphs with little effort and greatly helped the analysis phase.

From the second half of December 2012, test results started to be analyzed to be published and produce the conclusion presented by this document. **Section 4** covers *the outcome* of the research.

TEST TYPE	STAGE	BEGIN	END
<i>System Tests</i>	one	19/Nov/2012	22/Nov/2012
<i>Network Tests</i>	one	26/Nov/2012	6/Dec/2012
<i>Combined Tests</i>	one	5/Dec/2012	6/Dec/2012
<i>System Tests*</i>	two	8/Dec/2012	10/Dec/2012
<i>Network Tests</i>	two	11/Dec/2012	13/Dec/2012+
<i>Combined Tests**</i>	two	12/Dec/2012	13/Dec/2012+
A few additional system benchmarks were run for confirmation on 21/Dec/2012. * using "reduced" benchmarking suites, see Section 3 ** using pure load generation instead of system benchmarks, see Section 3 + with the exception of some confirmation tests executed until January 2013.			

Table 3: Overview Execution periods of Tests and Benchmarks

3. THE TECHNICAL SIDE

3.1. SECTION INTRODUCTION

This section is devoted to the technical side of this project . It extends Section 2 and contains more in-depth implementation details and information about solutions used throughout the research.

This project required a broad range of System and Network engineering skills centered on *Linux systems*. The most important activities were:

- *setting up the test infrastructure (the "POC lab") with the virtualization candidates;*
- *setting up system monitoring and timekeeping monitoring inclusive of a TTM node;*
- *technical troubleshooting and remote administration;*
- *choosing, building and testing system and network benchmarks and test tools;*
- *automated execution of benchmarks and tests;*
- *automated fetching of monitoring graphs and results.*

3.2. THE CHOSEN VIRTUALIZATION TECHNOLOGIES

This paragraph provides more insight on the virtualization technologies chosen as candidates for testing based on specific factors, as explained in Paragraph 2.6.1.

3.2.1. VMWARE vSPHERE HYPERVISOR 5.1 (ESXi)

The **VMware vSphere Hypervisor 5.1 (ESXi)** is a commercial product at the core of the range offered by *VMware, Inc.* Freely available to download and licensed for limited use, it could be used in production but only under determinate legal conditions.



The ESXi Hypervisor provides a full virtualization solution. Virtual Machines offer similar resources as a physical system, taking away potential service compatibility issues: the vast majority of system components is abstracted on top of the Hypervisor itself. The high level of abstraction may cause issues in terms of the lack of access to a "real" time source and possibly also in terms of performance.

A negative aspect of the ESXi solution could be that its graphical management interface is based on the Windows platform, not in use at GII and not very scalable: the free vSphere Hypervisor does not offer a solution to aggregate multiple nodes into one single management interface.

The use of the graphical interface can be reduced to a near zero by freely available command line solutions based on Linux:

- the *vMA* or *vSphere Management Assistant* (ref [23] in the Bibliography) a separate virtual machine devoted to management purposes.
- the *vCLI* package (ref [22]) a set of command line tools that can run on any local or remote Linux system to perform management activities.

VMware offers the possibility to automate remote deployment of ESXi on remote hardware, for example from NFS shared storage, in a way remembering the *kickstart* of a Linux distribution (ref [16]). The Hypervisor can be protected from unauthorized access by a new firewall feature (ref[19]).

Once the ESXi Hypervisor is installed, pre-existing virtual machines templates saved with the standard *OVF* format (*Open Virtualization Format*) can be imported on a running Hypervisor eventually using a command line tool named *ovftool* (ref [15]).

Default resources available when creating a new virtual machine can accommodate by default many needs without any additional setup. Certain configuration changes can happen without the guests needing a reboot.

Much documentation and support articles for VMware products are freely available online from www.vmware.org. The high level of adoption of VMware products increases the amount of information available on the Net. The free ESXi is not entitled for support, however support packages can be acquired from VMware.

The ESXi also offers a limited interface to third party management tools through *libvirt*, a framework for virtualization supporting the remote management of many different technologies.

The *OPS (Operations)* department at the RIPE NCC does already manage a virtualization infrastructure based on the vSphere family of products and on different versions of the VMware Hypervisors. In the GII department, at least one senior engineer has experience with VMware products.

3.2.2. OPENVZ (OPEN VIRTUOZZO)

OpenVZ (or *Open Virtuozzo*) is an open source product developed through a community effort and available through the GPL v2 (General Public License). It is sponsored by *Parallels Inc* (that does not offer support for the open source product) and used as a basis for one of their commercial products. The GPL licensing scheme leaves much freedom to use and tweak the product as needed, while the presence of a sponsoring entity might form an indirect warranty about the continued life of OpenVZ.

Still a "virtualization" solution, OpenVZ is technologically very different from most *hypervisors* present on the market. Similar to FreeBSD *jails*, OpenVZ isolates an entire system into "*containers*" (from there the term "*container virtualization*"). Each container has its own resources assigned, like memory space, number of CPU cores and network interfaces, that are bound to those of the host system. Files and resources of a container (same as process handles) are all openly available and shared with those of the host system, for example it is easy to kill a process running into a container from the virtualization host: a handy feature for some system engineers, a serious security flaw for others.



The host system of OpenVZ runs a *customized* version of the Linux kernel, named *vzkernel*, maintained and made available by OpenVZ itself with download and setup instructions for the most known Linux distributions. In particular the OpenVZ kernel and tools packages are integrated within *Debian Linux* but are not included in RedHat based Linux distributions. *RedHat or CentOS users* just need to add the OpenVZ repository to their system. This research tested the 64bit stable *vzkernel* version: *2.6.32-042stab065.3*.

Lacking a full hypervisor, container virtualization (also known as *OS level virtualization*) cannot abstract all system resources therefore a container hosted by a Linux system may only run Linux, but *not necessarily the same Linux distribution of the host*.

The positive aspect of system containers is that they can supposedly enjoy near native access to the host system resources like disk I/O, memory and CPU. In fact, the *vzkernel* does perform some control of a number of resources that are not made directly available to the containers. One example is time: a container is bound to the timekeeping of its host and should not perform timekeeping on its own nor is permitted to do that by default, to avoid containers and host to compete while updating the same time resources.

OpenVZ is fully based on Linux and very easily manageable by a set of own command line tools. It can also be interfaced by third party management tools by using *libvirt*, however the standard version of *libvirt* provided by CentOS 6.3 does not seem to be compiled with support for *OpenVZ*. Extensive and clear documentation for the entire platform is to be found in the OpenVZ wiki: wiki.openvz.org.


Deployment of a container happens in a very simple way through *OS templates* containing an entire system. Using the *vzctl* tool it is possible to import and run a template into a container in less than ten commands. OS templates for standard distributions are available but customized OS templates can be made for easy and scalable deployment of different services on multiple nodes.

There is no direct experience with this specific technology in the GII department; however, the *FreeBSD jail* solution, based on a similar concept, has been used.

3.2.3. KVM (KERNEL BASED VIRTUALIZATION)

KVM or *Kernel-based Virtualization* does not form an entire virtualization solution but is made of different components. At the core of KVM there are two modules included in the standard *Linux Kernel* since version 2.6.20. Those modules provides access to a series of resources on which a virtual

machine can be run by certain user-level programs, by either using the *CPU extensions for virtualization* offered by most modern processors or through a software CPU emulator. The software named *qemu* is used to run the Virtual Machines using the KVM interface.

KVM offers full virtualization through the kernel modules directly hooked to the system. Its closeness to the inner kernel structures allows using *paravirtualized* drivers for example for I/O and networking. With  paravirtualization, the abstraction layer between virtualized and real resources is much thinner, potentially guaranteeing better performance. The feature that made KVM a good candidate was the presence of a paravirtualized time driver named "*kvm-clock*", offering on paper a more reliable clock source than fully virtualized solutions can generally do.

The components of the KVM infrastructure are *open source* and fall under a combination of licenses based on the GPL license.

RedHat Inc backs the KVM solution as a full-fledged infrastructure for virtualization and maintains a series of graphical and command line tools on its Linux distributions to manage it. Many of those tools are interfaced to KVM through *libvirt*, in fact KVM is fully accessible through *libvirt based tools*. Management tools present in RedHat Linux and derivate distributions like CentOS may be replaced by different tools in other distributions like *Debian Linux*. Probably the most important command line utility for easy and complete management of KVM on RedHat based systems is the one named *virsh*.



RedHat makes extensive information available about KVM, in particular through its *Virtualization Administration Guide* (ref [36]). Other information sources are spread across the Internet, there included some mailing lists. Being part of the standard Linux Kernel, support for the KVM modules happens through the usual development channels of the kernel itself. Generally speaking, each component of KVM has its own development community and support channels, which initially can be confusing in case support is needed or a bug must be filed for example.

KVM can be at the same time heaven or hell for system engineers in terms of scalability. Being fully integrated with Linux, a KVM infrastructure offers freedom and flexibility. The drawback is that its implementation requires much expert system engineering work on the host.

For example, *KVM* virtual machines can be stored into disk images, physical partitions or LVM logical disks, but any of them must be configured and prepared either manually, during the OS install or by 3rd party management tools like *cfengine* or *puppet*. This is good for integration with complex infrastructures where everything can be automated, but it moves the pressure for a well-functioning solution from the *maker* of that solution to the engineer who is implementing it. That marks a strong difference with a commercial solution (like *ESXi*) where a set of standardized choices are offered and are easily "up and running". In that extent, even OpenVZ offers a level of automation slightly better than KVM because OpenVZ is offered as an all-in-one solution versus the modular nature of KVM.

When using the standard RedHat Enterprise or CentOS Linux distributions, however, installing a few "package groups" is practically enough to install the many packages and dependencies needed by a KVM infrastructure:

```
yum groupinstall "Virtualization Platform" "Virtualization Tools"
```

Deployment of virtual machines into a KVM infrastructure is possible in several ways. Importing, exporting, making a snapshot of a VM are possible using standard Linux tools (like the *snapshot* functionalities offered by LVM) . Virtual Machines templates in OVF format can also be imported.

There is limited direct experience with KVM within the RIPE NCC. However that should not form a big obstacle because of its integration with standard Linux tools and practices.

3.2.4. THE EXCLUDED VIRTUALIZATION TECHNOLOGIES

Due to time and resource limitations, some very good alternative candidates had to be excluded from this research.

LXC, offering OS level virtualization like OpenVZ but *fully integrated in the standard Linux kernel* like KVM could have offered an alternative solution for system containers. The main reason *OpenVZ* was preferred is that from some reports and from the information available, the LXC solution, however successfully used in some cases, appeared *not mature enough*.

Once born as a research project in England and later acquired by *Citrix, Inc*, but still available as opensource, *Xen* is an hypervisor running on Linux that could have been an excellent alternative to KVM. Like OpenVZ, Xen needs its customized kernel to run. Like KVM it has a number of paravirtualized resources, but its code is less tightly integrated with the standard Linux kernel code, however efforts are being made in that direction.

The RIPE NCC, now mostly using solutions from VMware, has had past experiences with Xen that were not always very positive. Information about the maturity and stability of the opensource Xen were questionable, and there seemed to be interest in GII to test a different technology, so KVM was chosen.

3.3. SETTING UP THE POC LAB

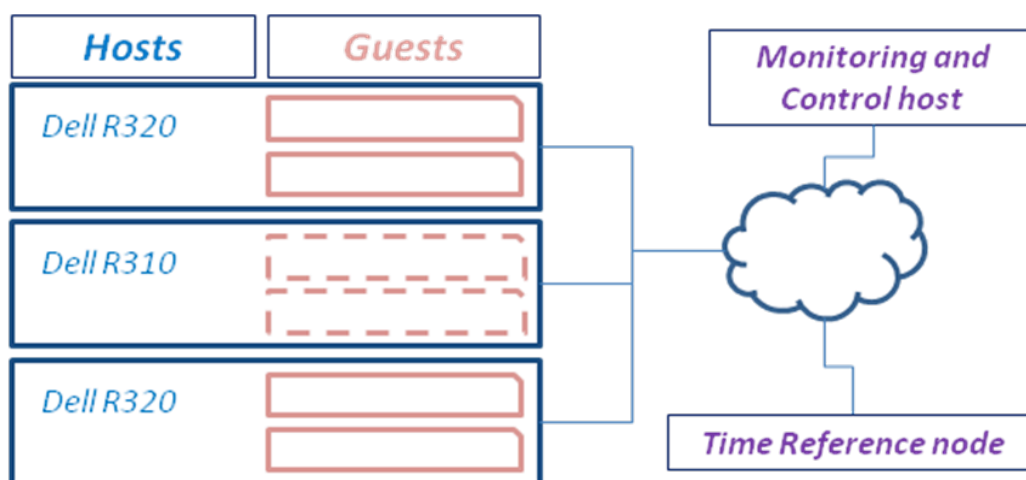
A reliable test setup is very important to make sure the outcome of a research can be relied upon. The following paragraphs describe how the testing infrastructure for this research (also named *POC lab*) and its monitoring were set up.

3.3.1. POC LAB HARDWARE AND VIRTUALIZED GUESTS

The POC lab was installed in a test space at the premises of RIPE NCC in Amsterdam. It was built using Dell rack-mounted servers:

- *Three servers* to host the virtualization candidates (named *colibri*, *robin* and *nightingale*). Each of them would host two identical guests on it, virtual machines or containers.
- *A fourth server* (named *bat*) for monitoring, control, collection and elaboration of results.
- *A fifth server*, named *tt999* and installed like a TTM (test traffic measurement) node was added as a reliable reference for time monitoring.

Figure 2: schematic view of the test infrastructure



Two servers hosting the virtualization candidates were *Dell Poweredge R320* servers with the same hardware specification as for the *Atlas Anchor nodes* (except for a lower amount of RAM memory): *6 CPU cores at 1.9 Ghz and 8Gb RAM*.

The third server was a lower end *Dell Poweredge R310* with *4 CPU cores at 2.4 Ghz and 8Gb RAM*. Reason for using different hardware was the lack of extra R320 hardware available and the need to perform as many tests as possible in parallel.

OpenVZ was running on R310 hardware during the first stage of tests, this made benchmark and test results not comparable among virtualization candidates during that stage but only *within* the candidates, to test resource separation.

It was assumed that using different hardware in stage 1 did not invalidate any of the tests that were executed. *Baselines* were made so that each virtualization candidate had an own reference for the "expected" performance.

The virtualized guests on each of the candidates were identical and were assigned the same resources *in each stage of testing*, according to this scheme:

- In *stage 1*, each guest had 2 CPUs (visible as 2 cores) and 2GB memory assigned
- In *stage 2*, each guest had 3 CPU (visible as 3 cores) and 4GB of memory assigned. This to better test resource separation when making full use of the physical resources.

All guests were allocated 4Gb swap and about 64Gb disk space each:

- In *VMware thin provisioned* local storage was used (the default choice).
- In *OpenVZ*, container files were on a separate LVM partition and shared the *ext4* file system with the host. In *OpenVZ* the swap space is virtual and managed by the kernel.
- In *KVM* one LVM Logical Disk was created on the virtualization host for each of the guests. The guests were *not* using LVM themselves.

3.3.2. NETWORK CONNECTIVITY

After consultation, this project was assigned consecutive network space in a VLAN named *NEW SERVICE*. The *NEW SERVICE* VLAN hosts much of the backend infrastructure of the *Science Division*, is broadly accessible, allows Internet access but is protected by a firewall.

All servers part of the POC lab shared the same VLAN and were connected to the same network switch, a *Foundry x424*. Having the time monitoring node in the same network space, with an average *round trip time (RTT)* of about 200 μ s, proved to be essential for precise time monitoring.

Using a fully separated VLAN might have been a better choice but was not possible. A shared VLAN made the POC LAB subject to network broadcasts within that network segment, however the use of a switch isolated the servers from unwanted non-directed traffic. This setup was assumed to be enough for scope and realism of this research.

All physical servers hosting a virtualization candidate had two built-in *network interfaces (NICs)*. Both interfaces were connected to the same switch on the same VLAN. For the majority of tests in stage 1 all virtualized guests were using just one physical network interface: the same as their hosts.

By the current design, the RIPE NCC box will be connected at remote locations by using one network interface only. On request of the manager GII and to test more implementation scenarios, in

stage 1 the second network interface was configured to carry the traffic of one of the guests for some special network tests.

In stage 2 the second network interface was used to separate (time) monitoring from test traffic.

In the VMware ESXi, the network interfaces of a VM are bound to the desired network interface with a click in the management interface or with a one-liner from the command line. The change does *not* require to reboot the guests and instantly transfers the traffic flows from one interface to another (depending on the Spanning Tree configuration on the switch).

Both in OpenVZ and KVM virtual Ethernet interfaces are automatically added to the system for each guest running. The *bridging* facilities of Linux can be used to bind those virtual network interfaces to the desired physical interface, either manually using *brctl* or automatically using the network configuration scripts (under */etc/sysconfig/network* in a RedHat/CentOS Linux distribution).

To be more precise, OpenVZ containers use by default a slow "venet" point-to-point interface to route the traffic from the host to the containers. This interface is fully controlled by the host including address assignment, has limited IPv6 support and cannot be modified from a container, where it is assigned a non-default name. Such a setup violated the service requirement for a virtualized resources and was not used.

Virtual Ethernet Devices (named "veth") were configured on OpenVZ containers, visible on a container as *normal network interfaces* and assigned default names (eth0, etc). Such 'real' interfaces are independent from the virtualization host where they only need to be bridged to a physical interface.

In KVM, the *paravirtualized* driver *virtio* was used for the network interfaces, promising better performance than fully virtualized interfaces. VMware VMs used by default a high performance network driver names *VMXNET3: also a paravirtualized driver*. Paravirtualization is a field of interest for VMware as well.

3.3.3. SYSTEM DEPLOYMENT AND THE DRAC CHALLENGE

For remote management and console, virtualization servers in the POC lab mounted a "Dell Remote Access Controller" (*iDRAC Enterprise version 7*), fitted with an vFlash SD card usable as local storage.

The server console can be accessed and used through either the DRAC web interface or ssh. In the original intentions of GII engineers, the combination of *DRAC and vFlash* could be used to install the operating system on the remote nodes. An ISO image could be uploaded to the vFlash, and the servers could be booted and installed using that image. RIS boxes had been installed locally using that method.

The DRAC+vFlash solution came out to be impractical with remote nodes: upload of ISO images through the web interface has proven to be unreliable and slow, unless done from a local network.

Nonetheless the two CentOS systems and the ESXi hypervisor needed by this research were installed using that method, to respect the original intentions of GII engineers.

Further research during the *Atlas Anchors pilot phase* found a better way to install a Linux system on a remote node. First by putting the ISO install image on an accessible network share, to upload it to the vFlash by using appropriate command line tools provided by Dell; then use the *ssh* facility to obtain a console of the server and proceed with the setup. The web interface of the DRAC uses java for the system console and often presents incompatibilities with *non-Windows* systems.

It remains unclear whether a similar solution can be used if the ESXi hypervisor has to be installed instead, as in ref [16] in the bibliography.

3.3.4. VMWARE VSPHERE 5.1 AND DELL R320 HARDWARE

The standard VMware ESXi Hypervisor 5.1 ISO image, available for download from www.vmware.com when setting up the POC lab, did not support Dell R320 built-in network interface. That is still the case at the moment of writing.

A custom image of the ESXi 5.1 with the missing support for the network interface is made available from Dell in the *Driver Downloads* section of their support site reachable from www.dell.com. That ESXi 5.1 customized image is the one installed in the POC lab. VMware might add the missing driver in future versions of the official ESXi image.

3.3.5. OPERATING SYSTEM SETUP

The first step when designing the POC lab was to know and respect the RIPE NCC operative standards. Examples of activities involving the local infrastructure were: agreeing on the best network VLAN, asking specific firewall rules, choosing host and domain names and insert them properly in the *ripe.net* domain. Or decide the best way to install a Linux host. Much cooperation about these matters happened with the *OPS* department and less often with *GII* engineers.

The RIPE NCC infrastructure offers PXE automated setup of CentOS Linux and uses the system configuration and management tool *cfengine*. These options were initially considered for the POC lab to have test systems as close to RIPE NCC standards as possible. After a number of trials, conclusion was that the research could provide cleaner results *without* using the overhead added by that automation. In addition, the future RIPE NCC box would not use that infrastructure in its current form and it was not among the scopes of this project to make a *cfengine* configuration for the RIPE NCC box.

All virtualization hosts, virtual machines and containers in the POC lab (except the host intended to run the ESXi VMware Hypervisor) were installed with a basic 64bit setup of the *Linux CentOS 6.3 distribution*, by using its official "*network install*" ISO image. In OpenVZ the latest official CentOS 6.3 template had to be used instead of the ISO image. Under VMware the *vmware tools* were also installed in the guests.

Once installed, all POC lab servers, hosts and guests were made remotely manageable through shared *ssh* keys. A number of shell scripts helped the automation of several operations throughout the POC lab during the entire project, included package and benchmarks install, configuration, fetching of logfiles and test results and so on. Most operations were centralized on the monitoring host *bat*.

A full upgrade of all packages was executed on all Linux hosts and guests at the end of October 2012 (before the test had begun), after which packages installed, configurations and enabled system services were frozen and kept for as much as possible equal across all servers, to guarantee the running of benchmark and tests under the same conditions.

This research used the OpenVZ *RHEL6-based vzkernel* version 2.6.32-042stab063.2; KVM host and guest and both VMware virtualized guests used the CentOS kernel version: 2.6.32-279.11.1.el6.

3.3.6. NTP CONFIGURATION

Timekeeping deserved special attention in this project, becoming the subject of another *research into a research* that provided a plethora of interesting information that unfortunately cannot be all presented into this document.

For the sake of this document, suffice to say that all PC compatible hardware uses a *clock source* to keep its time synchronized in hardware.

In early PC the clock source was the archaic CMOS clock powered by a battery when a system was off. Today there are more modern clock sources, not least the HPET (High Precision Event Timer) in the chipset and the *ACPI PM* (Power Management) counter. On Linux systems the chosen default clock source is the *TSC or Time Stamp Counter*, that counts the number of clock cycles since the last boot. Modern multicore system with dynamic clock rated provide a *constant tsc* whose rate remains fixed if the core rates vary, to prevent time drift due to sudden clock variations.

The *network time protocol (NTP)* played an important role into *time monitoring*, as paragraph 3.5 will describe, but a good *NTP* configuration was also essential to keep the right time on both virtualization hosts and guests, like some graphs in the *Results-Attachments* document will show.

Virtualized guests all have a virtualized *constant tsc* and use that as standard clock source. But how good would timekeeping be when only using that clock source and no NTP?

A test has been run by leaving host and virtualization guests running with the NTP daemon *ntpd* inactive for several hours (i.e. only querying the inactive server used for monitoring). The resulting graphs (in the *Results-Attachments* document) have shown a time skew incompatible with the requirements of this research. NTP or any better time synchronization method *must be used on production servers where the time resource is critical*.

VMware also offers a facility to periodically synchronize its guests to the time of the hypervisor host through the VMware tools at regular intervals, but VMware itself in a number of sources suggests to rather to use NTP for better timekeeping on Linux systems (ref. [17] in the bibliography).

Each virtualization host and guest in the POC lab ran the *ntp* daemon with a configuration similar to that of new GII servers provisioned by the *cfengine* management tool. That configuration contained *TTM* servers located in the region of Amsterdam. As a reminder *TTM* nodes with their built-in GPS time reference offer a very precise *stratum 1* clock source with maximum offset variations below 50 μ sec.

The NTP configuration used in the POC lab would fit well with the generic case of a RIPE NCC box, since the POC lab is located in Amsterdam. GII might want to carefully evaluate what can be the best standard NTP configuration for remote *Atlas Anchor nodes* or *RIPE NCC boxes* and especially when *TTM* nodes will start to be decommissioned.

The two OpenVZ containers were the only necessary exception to the standard NTP configuration. By default, OpenVZ does not allow to run *ntp* from a container, unless configured to do so. Running the NTP daemon from within a container *and* on the virtualization host is not recommended: containers share the same clock source as their host and multiple NTP daemons would end up to compete for timekeeping.

OpenVZ containers have been configured to allow the run of NTP, but *without any active peer* except the inactive one used for *monitoring* purposes, as explained in paragraph 3.5. In this way the NTP daemon on the container would not conflict with the one on the OpenVZ host. Should OpenVZ be used in production, NTP does not need to run at all into a container.

3.3.7. THE KVM-CLOCK PARAVIRTUALIZED CLOCKSOURCE

At the beginning of this research much expectations were posed on a relatively new paravirtualized clock source offered to its guests by KVM named *kvm-clock*. This clock source would provide a much more stable time reference than the virtualized *tsc*. According to some sources, notably SuSE, in presence of the *kvm-clock* clocksource NTP should not be used for active timekeeping (same as for OpenVZ) to prevent conflicting timekeeping between virtualization host and guests (ref. [41]).

kvm-clock is active by default and was of course tested on both KVM guests in the POC lab when they were installed. Unfortunately, its performance was largely *not* up to the expectations.

When *kvm-clock* was used, at least one of the two KVM guests show through the monitoring very strange time drift patterns, every time different and hardly ever regular, always failing at maintaining any accurate timekeeping. The 'misbehaving' guest seemed to be the one booted as last. The other guest did manage to maintain good time with sub-millisecond stability, but often showing incidental 'big' offset jumps around 300-500 μ s (compared to 100-200 μ s variations on VMware guests).

Several attempts have been made to understand the source of this unstable behavior. Similar behavior was seen with both an earlier kernel version (2.6.32-279.9.1) and the most recent one (2.6.32-279.11.1).

Configurations and settings were checked and others were attempted: NTP servers for time synchronization have been enabled at some point in the KVM guests as RedHat suggests (ref. [42]). In that case a fight was visible in time monitoring graphs, with *kvm-clock* producing wrong time skew and the NTP daemon trying to correct that, making matters worse.

Not all graphs for all trials are available or have been indexed because this issue happened at the early stages of the project when the entire monitoring was not yet in place, but the "*baselines*" section of the *Results-Attachments* document show both cases of *kvm-clock* being observed *with* and *without* NTP peers active for twelve hours (NTP was always active on the virtualization host, though).


This research concluded to the best of its findings that the *kvm-clock* paravirtualized clock source is at least not yet mature enough to be used, or incompatible in some way with either the kernel versions tried or the hardware used.

To continue the research, *kvm-clock* was permanently *disabled* in KVM guests, by using the '*no-kvmclock*' kernel option at boot. KVM guests reverted to use the virtualized *constant_tsc* clock source, just like VMware guests did. Since *kvm-clock* was disabled, both KVM guests (with NTP now permanently active) did show a relatively more stable timekeeping with a similar order of magnitude as that of VMware guests, however KVM guests apparently kept showing broader offset variation trends.

3.4. SYSTEM AND NETWORK RESOURCES MONITORING

The POC lab server named 'bat' was the center of all test, monitoring and data processing activities.

The status but above all the performance of system and network resources was subject to passive monitoring by polling *snmp* and *nrpe* sources on all POC lab hosts and guests and to active monitoring by sending them DNS queries and by generating ping bursts *from and to* them. The tools or software packages used for monitoring were all open source:

- "*cacti*", the most important source of information, graphing all system and network load statistics with a polling interval of one minute, RRD data retention up to one year and the use of the advanced poller named '*spine*'. Extra data templates of *cacti* were implemented, to better watch disk I/O transfer rates, NTP, detailed CPU, I/O and memory statistics produced by *sar* and *iostat* tools that ran on the systems and were queried through *nrpe*. 
- *Observium*, a fully automated passive monitoring tool like *cacti*: it offered a second observation point watching the most important metrics in parallel with *cacti*.
- *Smokeying* performed *bi-directional network latency monitoring* with a poll interval of one minute and final RRD data retention of two months. *Smokeying* was configured with three kinds of tests to execute through the primary network interface:

- ping bursts produced by *fping* were sent at regular intervals to hosts and guests;
- all hosts and guests sent automatic ping bursts to the time monitoring node;
- equal DNS queries were made from the monitoring server to a simple caching DNS server installed on all guests and hosts of the POC lab.

Ping tests allowed to observe bidirectional round trip times (*RTT*) and packet loss variation on the network links. DNS checks allowed to see any change in DNS response times.



Of the entire monitoring setup Smokeping offered the most insightful graphs, especially important to observe resource separation and the ability of the different technologies to cope with high load on the network. The graphs provided by cacti, were useful as a figure of the system load levels, the disk I/O and the network interface throughput reached during tests and benchmarks.

3.4.1. MONITORING LIMITATIONS

The system monitoring had some clear limitations. The polling interval of 1 minute in *cacti* and the test interval of 1 minute in *Smokeping* could of course only provide a resolution limited to one minute. That is the reason why cacti could not be the primary method to monitor the stability of time.

One minute was actually a ceiling: cacti had more than 250 target items to poll and its standard poller coped badly with them when some timeout occurred. The multithreaded *spine* poller improved the fetching of statistics only after proper fine-tuning. The *Smokeping* engine written in perl, in combination with the *apache* webserver had severe troubles and high memory requirements when the RRD data retention period was longer than two months, with 1 minutes data testing intervals and the retention of all results. Using *fastcgi* slightly improved its functionality.

Another limitation of the monitoring was in its empirical nature: all monitoring results were necessarily approximations of the real picture, not scientifically precise measurements. Such approximation did suffice to come to relevant conclusions and give answer to the main scope of this research, but it fell short when more precise data analysis was necessary.

3.5. MEASURING TIMEKEEPING STABILITY

Another essential challenge of this project was to figure out a way to monitor the time stability of any virtualization host or guest with sub millisecond precision and with the lowest possible error rate.

One possibility could be to use a GPS antenna connected to each virtualization host, in a way similar to Test Traffic Measurement nodes, and then refer to the hosts as time reference. This idea was not applicable for many reasons.

Using an externally powered GPS providing a stable pulse through the serial port would not be a good idea: that pulse would keep the host in sync, but not the guests. A virtualization guest cannot use its own host as time reference, to avoid a dangerous circular reference (ref. [17] and others).

The NTP (*network time protocol*) daemon *ntpd* offered the solution. I will present some elements of the inner functionality of NTP time synchronization (quite briefly due to scope/space limitations of this document), to describe how time monitoring was implemented.

The algorithms implemented in the NTP daemon, *ntpd*, selects one *preferred server* among the ones configured and queries it about once a minute by default, dynamically changing the query intervals or selectively preferring another server, depending on a number of factors. The returned timestamps of multiple queries are evaluated to calculate the best possible time offset; the system time is then slowly corrected as necessary.

When enabled, the standard logging of the NTP daemon reports the calculated time offset or time skew of each of the configured servers in the *peerstats logfile*. A query performed by common monitoring tools or using the *ntpq* tool reports that offset, that is: the estimated time offsets after the NTP algorithms evaluated multiple raw queries to its servers.

The NTP daemon can also save unfiltered results of each single NTP query in the *rawstats logfile*. Raw NTP queries, one query reported on each line of the logfile, contain among other fields the *peer address* and *four timestamps*:

- The *ORIGINATE timestamp* (OT) set by the sender when the query is sent.
- The *RECEIVE timestamp* (RT) set by the receiver when the query is received.
- The *TRANSMIT timestamp* (TT) set by the receiver when the answer is sent.
- The *DESTINATION timestamp* (DT) set by the sender when the answer is received.

The NTP daemon elaborates those timestamps to calculate delay to the ntp server, time offset and jitter or the standard deviation between offset reads, and then writes those values in the *peerstats log*, as described earlier.

According to ref. [11], offset and delay can be calculated by using the following formulas:

$\text{offset} = \frac{(RT - OT) + (TT - DT)}{2}$	$\text{delay} = (DT - OT) - (TT - RT)$
---	--

To perform time monitoring for this research, each host or guest in the POC lab ran an NTP daemon performing raw query logging.

One Test Traffic (TTM) box installed in the same network as the POC lab was set as a *noselect* server in the configuration of the *NTP daemons*. With the *noselect* option that server was never chosen for time synchronization. Time synchronization was done by using *selectable* ("active") NTP servers or not done at all if no other servers were specified (*for example in OpenVZ guests, see paragraph 3.3.2*).

The minimum and maximum polling interval for that *noselect* server were set to the minimum possible (3^2 seconds) so to query that clock source about every 8 seconds. The rest of the NTP configuration remained unchanged.

An excerpt of the NTP configuration present on all hosts and guests for time monitoring is published below.

Excerpt ntp.conf for time monitoring


```
# Never panic
tinker panic 0

# One stratum 1 source reference in the form of one TTM box
# for statistic collection only
server tt999.ripe.net noselect minpoll 3 maxpoll 3

# Enable writing of statistics records.
statistics clockstats cryptostats loopstats peerstats
rawstats
```

Automatic scheduled jobs on the monitoring server *bat* fetched and saved each night the daily *rawstats* logfiles from each POC lab server. A *conversion* script ran for each host, extracting the query answers from the test traffic box intended for monitoring, converting the time format from UTC to readable values, then calculating *offset and delay* with the formulas above.

Converted time, offset and delay were saved in a file. Statistics and a 24 hours graph for that day were generated based on that and stored in a tree of folders accessible through a web browser.

 A set of graphing scripts and templates that I wrote for the Software *gnuplot* could be run either manually, or automatically by the graph generator scripts described in paragraph 3.7, to produce high resolution graphs of the timekeeping for a certain period and with a certain scale, aggregating guests and hosts of each virtualization candidate in one graph.

In addition to *rawstats* time monitoring, NTP monitoring of much lower resolution was performed using *cacti* and sending *ntpq queries* to that same TTM node used in the monitoring through *rawstats*. The graphs generated through *cacti* were often used as reference to confirm the results produced by the *rawstatS-based* graphs.

3.5.1. TIMEKEEPING QUIRKS AND POLLING INTERVAL LIMITATION

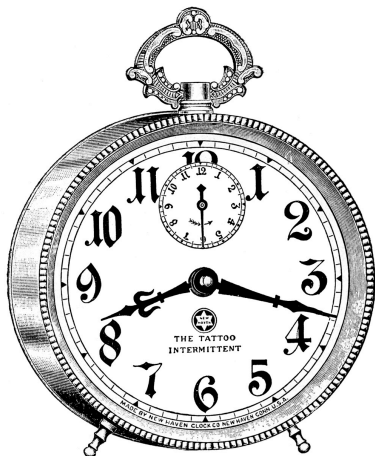
All time monitoring graphs in the Results-Attachments document are in fact formed by two graphs, the one above shows the offset variation or time drift registered. The one below the corresponding delay to the time reference node, more or less corresponding to the *Round Trip Time*.

The delay in time monitoring was an important measure of two factors: the precision rate of the monitoring itself and some insight about the origin of the time skew.

The precision rate of the offset variation is given by the average delay: by a round-trip delay of 200 μsec , the time offset was correct by a possible variation of $\pm 200 \mu\text{sec}$. Enough for this research seeking time precision below 1000 μsec (1ms). In fact, the first TTM node for time monitoring (*tt97.ripe.net*) resided in a different VLAN than the POC and had an average delay of about 1100 μsec ! That was certainly not good enough to measure sub millisecond time accuracy and was replaced by *tt999.ripe.net* installed in the same VLAN as the POC.

When looking at a variation in the offset, one should also look at the corresponding delay at the same moment: if a 1ms offset saw a correspondent 2ms delay (round trip, thus double time): that meant that the time drift could be related to some "slowness" to reach the monitoring server, not necessarily indicate "real" drift of the system clock.

A higher delay could be caused by too much CPU or I/O load on the host, or by network buffers full or by some peculiar network disturbance along the path to the monitoring node.



There was no way to establish each time the real cause of a peak in delay with a correspondent offset and if that indicated 'real' time skew or was just due to 'something' slowing down a few NTP queries. If not using rawstats statistics, the NTP algorithms would have filtered out the *peak exceptions* and come out with a mean value. But in that case we would probably draw *what NTP thought* the real offset was, i.e. an approximation.

Those *peaks* in time monitoring invited to observe time drift more as a continuous function. For the way timekeeping works in a Linux system, subsequent calls are made to the *adjtimex()* function, that reads the clock source and performs any necessary adjustment. NTP works on top of the system time and calls the function *adjtime()* when has to make a change smaller than 2145 seconds. That change happens by a gradual slew of the clock (according to an unconfirmed source: 0.5ms per second at most, in Linux): *not suddenly*.

By design, NTP would not suddenly force a big jump in time that could cause instability (calling the function *settimeofday()*) unless having a considered good reason to do so.

One-time peaks with correspondent delay could then better be looked at critically ("can this be a query facing a loaded network stack?") to rather observe the *trends* of timekeeping. *Not single peaks, but many peaks after each other or a steady variation of the offset would more certainly indicate time drift*. Even the time baseline of a physical host (in *Results-Attachments*) shows occasional small peaks.

One of the critics to the time monitoring implemented for this research was that the NTP daemon had a *minimum interval between queries* of 8 seconds (the shortest allowed, set through the *minpoll*

and *maxpoll* configuration options), so that nobody could be sure of what happened during those 8 seconds and if the time remained stable.

Statistically, 8 seconds polling interval meant as many as 10800 queries in 24 hours or 450 in an hour, giving good grounds for the probability that no time skew had happened during the 8 seconds. Also based on the previous remarks by which the system clock would not incur into big jumps at once and then jump back to the original value within such a short time frame.

Under normal circumstances, in a Linux system the maximum possible 'real' time skew during 8 seconds should top 4ms at the most ($0.5 * 8$). This is presumably a hard limit of this monitoring method, together with the level of uncertainty given by one-time delay increase and relative offset peaks. To be sure of the most absolute adherence to the requirements NTP should have been made capable to send an ntp query every 2 seconds at the most and always have an impossible flat delay, or another method for time monitoring had to be invented.

3.5.2. SHARED NETWORK LIMITATION AND USE OF SEPARATE INTERFACES.

Another limitation of the original time monitoring implementation in this research was due to the fact that network communication to the time monitoring node had to be done on the same network interface as any other traffic, and then was subject to network traffic on that interface. Reason for that setup was the requirement that each guest should only have one network interface (2.5.2).

Consequence of the limitation was that *time monitoring was not reliable during network and combined tests in stage 1* because communication to the time monitoring node was too disrupted by the network load to guarantee consistent results.

During stage 2 of testing that limitation was partially taken away: monitoring traffic including time monitoring was redirected through a second interface using private IP addresses. Virtualized guests had this interface bound to the second physical interface of their virtualization host. A temporary exception to the service requirement of the *Atlas software probe* to only have one interface active on the system where it runs.

By using separate interfaces data flows were then separated so that *time monitoring was made more reliable also for network related tests in stage 2*, an important breakthrough that allowed to see interesting additional results, but added certain challenges to data analysis.

Time did not allow to fully separate the traffic flows, configure a new VLAN and the second interface in the time monitoring node: hosts and guests were talking to the time monitoring node using different interfaces and different IP addresses but *still on the same VLAN*. Looking at the 24 hours time baseline for a hardware host with no virtualization has given enough grounds to consider the drawbacks of sharing the same VLAN *negligible* enough for this research..

3.6. MEASURING RESOURCE SEPARATION

Resource Separation and Availability was a required feature of a virtualization candidate for the RIPE NCC box. In the intentions of this research, that requirement could be tested using these steps:

- Find a method to produce *repeatable load* on guests and hosts while allowing to measure the performance achieved.
- Produce load on each host and each type of guest alone and measure the available processing power available (or disk I/O performance, or RAM speed), to have a *baseline* of the expected performance.
- Produce load and measure the available processing power available (or disk I/O performance, or RAM speed) under a number of relevant *use cases* of competition on resources.
- Compare the results *within* each virtualization candidate between the *baseline* and the other use cases to see how the competition on resources had affected the performance of one or both players in each use case.

The extent of the services to run on a RIPE NCC box was not fully defined at project start (except for some basic requirements). To embrace the maximum number of possibilities and give widely usable answers the benchmarks executed during this research tried to remain *generic* for as much as possible, focusing on the *technical requirements* of the virtualization candidates but not on specific RIPE NCC services.

On the other hand the project took into consideration some specific factors, for example the assumption that given the hardware specification provided and the candidate services expected, A RIPE NCC box would generally not overcommit its CPU or memory resources, because it would run just 2 or 3 services in its first year of operation or longer. This helped to decide the sizing of the guests and excluded resource overcommit use cases to focus on resource *sharing* or *fair separation* instead.

All kinds of tests executed had a *name tag* assigned that was used throughout the research to avoid confusion. Test names and all use cases considered by this research are detailed in the *Results-Attachments* document .

3.6.1. TESTING THE VIRTUALIZATION HOST

Next to virtualization guests, on request of the GII manager this research had the extra scope to test and benchmark also *virtualization hosts* to see if they could fit the requirements to run production services. Certain services could then run on a guest, others (for example more time critical ones) could

be left to run in the virtualization host. For this extra scope a number of additional use cases were added.

Only OpenVZ and KVM offered an "usable" virtualization host to test. The hypervisor in ESXi 5.1 (unlike the older ESX hypervisors) cannot be considered like the full-fledged Linux virtualization host of *OpenVZ* and *KVM*. The *ESXi* hypervisor is a very stripped down system. It can be made remotely accessible through ssh (but no ssh keys allowed) or run a stripped-down ntpd daemon for example, but has very limited support for installing or building new packages and in many ways does not behave as a physical host. For example every time the ESXi management interface is used the clock of the hypervisor (independent from its guests) is automatically synchronized to the clock of the Windows machine running the management interface, that would often also reset its NTP configuration files!

Time statistics have been collected from the ESXi hypervisor host in an initial period and the time on the host (with the usual NTP configuration) always looked relatively unrelated to that of its guests. The ESXi host would not be included into the host use cases for testing and benchmarking.

3.6.2. SYSTEM BENCHMARKS AND LOAD GENERATION

Finding concrete and repeatable ways to test and compare CPU, Disk and memory performance in different use cases was no easy task. A number of variables and questions did play a role, like the *synchronization* of the load applied when two guests were tested simultaneously.

Two *Benchmarking packages* were chosen for the scope, those were:

- The **Phoronix test suite**, one of the most complete benchmarking frameworks for the *Linux* operating system freely available under the GPL license. The software, written in php and compatible with different Linux distributions, is able to automatically fetch all dependencies, download, compile and transparently run a wide range of benchmarks. It then automatically generates results web pages but is also able to manipulate and export the results.
- **Unixbench**, a "basic performance indicator of a UNIX-like system" to quote its description; freely available under the GPL v2 license and implementing a number of well known system benchmarks, like *Dhrystone* and *Whetstone* to test the CPU performance, derived from the ones used by the historical BYTE Magazine.



The "*Phoronix Test Suite*" (*PTS*) software is associated to the initiative of the site openbenchmarking.org to openly share benchmarking suites and results. A few specific test suites available on the site were chosen for this research. Together with the version 4.2.0m2 of the PTS software, these were installed, built and tested on each host and guest in the POC lab between October and November 2012:

- **pts/cpu**, of variable duration, for testing and load stressing the CPU resources

- **pts/disk**, of variable duration, for benchmarking disk I/O resources
- **pts/memory**, duration between 30 and 40 minutes, for benchmarking the memory

The *pts/memory* suite was also used in stage 2 when, in order to produce a smaller but more usable subsets of results, two customized suites replaced *pts/cpu* and *pts/disk*. Both of them with an average duration of an hour:

- **ripebox-cpu**
- **ripebox-disk**

Despite many efforts, not all benchmarks part of a suite did always run successfully. This can happen when running system benchmarks but could invalidate certain conclusions of this research when some tests were meant to run in parallel. Because of that all test suites have been tested before the 'official' runs: certain issues have been solved while other benchmarks that were too unpredictable were disabled.

In stage 2 an additional load generator was added: the opensource tool called *stress*, able to predictably generate high CPU, DISK or Memory load. The *stress* tool has been tested to find the best parameters that would not cause crashes in the POC lab. After that, a 20 minutes period of heavy and stable load generation (respectively for CPU, DISK or Memory) was automatically started after each of the benchmarking suites. This very extreme and entirely reproducible source of load has added interesting insights to the end results.

The *BYTE CPU benchmarks* always failed when run through the *Phoronix Test Suite* and had to be disabled from there. After stage 1, the impression was that some more "dry" metrics to compare CPU performance were missing. For that reason also the **Unixbench** software was installed in the POC lab during stage 2.

A compact, complete system benchmarking suite, *Unixbench* was run separately from the PTS and through a number of insightful use cases, providing an additional figure of system performance as well.

Description of the benchmarks part of the test suites, numeric results of all single benchmarks executed, comparative tables and remarkable graphs of the timekeeping during system tests are published in the *Results-Attachments* document. Results of System Benchmarks executed through the *Phoronix Test Suite* by this research have not been published on *openbenchmarking.org* due to a limitation imposed by the license of one of the virtualization candidates.

3.6.3. NETWORK TESTS

Testing a network is an entire field of study with potentially hundreds of variables. Many factors external to the host being tested can adversely influence the results and have to be studied.

Efforts were made by this research to minimize external factors and contribute to the reliability of network tests by building a consistent test infrastructure where even the CAT 5E cables used were the

same. Extra care has been put in checking that hosts and guests being tested were not generating other unwanted network traffic during the tests. *Baselines* have been made to observe the 'expected behavior' when necessary. That lead to the observation that *network flows can hardly be identical*.

Despite all efforts, the results produced by network and combined tests were and remain of empirical value. This research did not aim to prove which network interface or network stack was performing better, like it did for system resources.

Primary scope of network tests was to check *resource separation* of the network stack under different virtualization technologies. In addition to that: in some case the setup of stage 2 helped observing the impact of extreme network load when the timekeeping relies on NTP.

After some research, the packet generation tool *iperf* was chosen for network and combined tests. Freely available through a *BSD license*, *iperf* is able to generate load using either TCP or UDP protocols and also reports certain statistics when executed, like *bandwidth* and *packet loss*. The tool can generate bidirectional TCP or UDP packet flows through a server/client infrastructure.

For each network test, a *server/client pair* was needed. When two systems were involved in a test, two pairs were needed for a total of four systems involved. Because no extra hardware was available for testing, test pairs were formed in each stage using all hosts and guests of the POC lab. Also for this reasons network tests could only be executed sequentially, on one virtualization technology at a time. The network performance of two different pairs could not be exactly compared for small differences.

The *iperf* tool was implemented into an automated testing infrastructure I developed in order to generate two main types of data flows:

- a flow based on 10 **TCP** sockets handled by 10 threads, with the scope of filling up the data link (all network links were 1 Gb links);
- a 120Mbit **UDP** flow, to observe data loss and latency variations.

Those simple data flows were combined in several different use cases to be run among guests and hosts, in a similar way as system tests. That could give answer to several questions about network resource separation and beyond.

The vast majority of tests were executed in what is the current implementation of *Atlas Anchor Nodes* and was expected to be the initial implementation of the RIPE NCC box: *all virtualization hosts and guests sharing the same network interface*.

On the request of the GII manager and to envisage future implementation scenarios, three extra network tests looked at the case that *two network interface would be used for production traffic* in the RIPE NCC box. For that use case one of the guests was bound to the second physical interface on the host.

More details about the server/client pairs and extensive explanation of network tests use cases are published in the *Results-Attachments* document.

3.6.4. COMBINED TESTS

Combined tests were meant to observe the interaction between system and network stack in case of load; their most interesting contribution to this research was to observe the impact of combined system and network load on timekeeping and NTP performance, a target made possible in stage 2.

In stage 1 combined tests employed the same test types or test suites as in network and system tests: one system would run one benchmarking suite to generate load while the other would generate one of the two network data flows defined in the previous paragraph.

Combined tests, initially producing redundant results in line with those of System and Network tests, have proven essential for conclusions about timekeeping and to confirm earlier findings.

In stage 1 the load generated by benchmarking suites was not steady enough to have much impact on the network activity in the other VM. For this reason, system benchmarks were entirely replaced in stage 2 by the stable heavy load generated by the *stress* tool, respectively on CPU, DISK or Memory.

The *Results-Attachments* document explains use cases and combined tests that have been run.

3.7. AUTOMATED TEST SCHEDULING, DATA FETCHING AND GRAPH GENERATION

Network and Combined tests required extreme care in making sure they were running undisturbed, and that all monitoring metrics were collecting the needed information. Multiple trials had to be run before seeing stable results.

System benchmarks were manually scheduled on each guest or host using the *'at'* *system* utility. To run network and combined tests a layer of automation was developed instead, using shell-scripting language.

An input file on the controlling host *bat* would contain a complete schedule of all use cases to run in sequence, written according to a certain format. At specific times, a *"test run script"* would read the next use case to run and remotely invoke a *"test execution script"* in one of the POC hosts or guests.

The *"test execution script"*, in turn, would start all requested tests or benchmarks on the host where it was running, on its paired server in case of network tests and also on the other guest, in case of combined tests: all spawning *screens* in background. The *test execution script* also took care of logging the output of tests in the respective hosts or guests where they ran.

The automated test execution, perfected after extensive debugging, allowed to run several rounds of tests of fixed duration also during the night, dramatically cutting down the time needed, or almost doubling the time available to this project to run benchmarks and tests.

The "*test run script*" part of the automation infrastructure also had another scope: it translated the test schedule to another format associating test names and execution times, to allow the *automatic generation of graphs*.

The "*drawgraphs*" *graph generation script* for network tests and its enhanced version "*drawgraphs+*" for system and combined tests read test names, time and date of execution and then fetched or generated specific types of graphs for the specific hosts tested in each specific timeframe.

That script had three different sources:

- the *cacti* monitoring system
- the *smokeping* network latency monitoring
- *ntpd* rawstats from all hosts and guests (daily fetched and saved locally on the *bat* server): in this case "*drawgraphs*" called existing scripts that generated time monitoring graphs.

The same script finally saved the graphs in a directory tree structure ordered per test name and virtualization type. That tree, accessible through a web server, allowed immediate consultation of all system, network and time monitoring graphs of a specific test run by just knowing the test name.

A test scheduling file looked as simple as this excerpt, the scripts took care of all the rest:

Excerpt 1: test scheduling input file

```
NS5,KVM,test5+test6
tcp test5 tcp test6 300
NS6,VMware,test1+test2
tcp test1 udp test2
NS6,OpenVZ,test3+test4
tcp test3 udp test4
```

The "*test run script*" also took care to set the start time of tests to some minutes earlier so that the graphs would show the change from *idle* to the moment a test began. Another excerpt shows an example of the file for automatic generation of graphs it produced:

Excerpt 2: input file for automatic graph generation

```
NS5,KVM,test5,2012-12-06,08:55,2012-12-06,09:30,test6
NS6,VMware,test1,2012-12-06,09:25,2012-12-06,10:00,test2
NS6,OpenVZ,test3,2012-12-06,09:55,2012-12-06,10:30,test4
```

One full round of network tests in stage 1 produced as many as **654 graphs!** Certainly not all of them were necessary, but each of them could be easily consulted at any time speeding up the data analysis consistently.

4. OUTCOME AND ADVICE

4.1. SECTION SUMMARY

This section summarizes results and conclusions of this research. It also brings an advice about possible perspectives for the *RIPE NCC box*.

Detailed results, comparative graphs, monitoring and timekeeping graphs are published in the *Attachments-Results* document associated to this paper; implementation details, also covering some scalability aspects, are discussed in *Section 3* of this document.

Concerning the research outcome: each of the main technical requirements is treated separately, with a subsection per each virtualization candidate. Paragraph 4.7 considers all the results and advices about short term and long term perspectives.

This part of the document ends with a number of *side notes* and finally a list of *open questions* and subjects not treated by the research or to be further investigated.

4.2. RESEARCH OUTCOME

A global summary of the outcome of this research is visible in *Table 4* below.

Requirements	Stable Timekeeping	Resource Separation				Scalable Manageability	Version Tested
		CPU	DISK I/O	MEMORY	NETWORK		
Virtualization Technology							
VMware ESXi	Yellow	Green	Green	Green	Yellow	Green	5.1 / Dell custom image for R320
OpenVZ	Yellow	Green	Yellow	Green	Red	Green	vzkernel 2.6.32-042stab065.3
KVM	Red	Green	Yellow	Green	Red	Yellow	kernel 2.6.32-279.11.1, qemu 0.12.1.2-2.295

Table 4. Outcome Summary Table

Compliant to requirements	Conditionally Compliant	NOT Compliant to requirements
---------------------------	-------------------------	-------------------------------

All candidates did in general comply to the *System Resources Separation requirement* for the way it was observed by this research (no resource overcommit of Memory or CPU).

When a virtualization host is present (*OpenVZ and KVM*) it is recommended to not use the host for production services, because it often negatively affects the performance of the guests. This is especially true when Disk I/O is involved.

Network Resource Separation, by which the network stacks of the guests behave independently from each other, is only supported by the ESXi Hypervisor in its default configuration.

An interference of the Memory subsystem under load with the Network stack has to be solved or have a concrete workaround, for the ESXi to reach full compliance to resource separation.

The product from VMware offers the best scalability when looking at a long term perspective, and can be very resilient and require very little maintenance efforts, once installed. This is different than KVM, with its modular nature and high likelihood of customized solutions that may need to be implemented and maintained.

The "*accurate timekeeping*" requirement, for the way it was defined, has no unconditional winner. Carefully configured *NTP* with stable peers is the minimal condition for stable timekeeping, however a ceiling of "*1ms*" set during the research may be too strict to be respected at all times by a virtualization technology and maybe even by physical PC hardware.

4.3. ACCURATE TIMEKEEPING

Timekeeping represented a critical requirement of the research. It has historically been a weakness of hypervisor-based technologies, because of the very nature of full virtualization connected with the way clock synchronization works.

Despite all instruments employed, the complex monitoring and the results obtained, this research is not able to give a definitely proven verdict whether the clock in certain virtualization technologies can remain stable at all times. It certainly appeared worse than on a physical host.

This paragraph provides some indications and presents the most relevant issues with timekeeping encountered during this research. The decision to accept one technological solution versus the other, or none of them, depends on the level of compromise that can be accepted (see paragraph 4.7).

4.3.1. VMWARE VSPHERE HYPERVISOR (ESXi)

When supported by reachable and stable *Network Time Protocol* servers, VMware has maintained a very good trend of **stable timekeeping** throughout the research, for an *Hypervisor based* solution. For that reason it has also been kept under tight observation.

A few issues with clock stability can be seen in the *Results-Attachment* document, where many timekeeping graphs regarding VMware are shown. One of the first events happened during a brief I/O hiccup caused by heavy disk benchmarks, something that has affected all technologies tested at least once.

An ESXi guest may incur into an occasional offset peak, often lower than one millisecond, that could often be associated to other subsystems like I/O, Memory or Network being loaded, as it is shown by a corresponding delay in a time monitoring graph.

Stage 2 tests and in particular some of the last graphs shown in the *Results-Attachments* document have demonstrated the necessity of an ESXi guests to have a stable external clock source like NTP in order to maintain the clock accurate.

The biggest issue with timekeeping in VMware was the *resource interference between memory and network stack* (paragraph 4.4.4) provoking network delays and concrete time drift, an issue that needs further investigation.

A workaround or a technical solution to prevent the issue above are the minimal conditions to make the ESXi compliant to all requirements of this research.

4.3.2. OPENVZ

The container-based virtualization offered by OpenVZ is **conditionally compliant to the timekeeping requirement**. For its technical nature, *OpenVZ* guests have been expected to offer near native access to system resources, and time accuracy equivalent to their physical host: the latter was often the case during the vast majority of tests and benchmarks.

However, the timekeeping graphs during network tests and the time baseline without NTP active peers published in *Results-Attachments* have shown the reliance of OpenVZ to well configured external clock sources, as discussed in paragraph 4.3.4.

In OpenVZ only one instance (either the virtualization host or one of the guests) can have an NTP configuration with active servers for synchronization, to prevent conflicts. During this project the virtualization host was performing clock synchronization.

4.3.3. KVM

KVM or the *Kernel Based Virtualization* solution is *deemed not compliant with the timekeeping requirement*.

Chosen as virtualization candidate with big expectations, the *paravirtualized* clocksource offered by KVM and named "*kvm-clock*" should provide unprecedented time accuracy for fully virtualized hosts. It proved instead a source of time instability, as the baselines published in the *Results-Attachment* document show. Possibly a good technical approach, maybe not mature enough: one of the reasons to exclude this technology for the scopes of this project.

After failed trials with *kvm-clock*, the KVM implementation in the POC lab was setup with the standard virtualized *TSC* clocksource, producing an acceptable baseline.

The time accuracy trends for KVM during normal daily operation in the POC, something that the graphs for single tests fail to show, were not very stable. Producing a 24 hours time baseline required quite some efforts: any load on the systems seemed to impact the clock and should be avoided.

When system benchmarks were run, the disk subsystem using the paravirtualized *virtio* driver *interfered* with the timekeeping in virtualized guests causing more than occasional single *peaks* but also relatively structural time drift. If not a purely "timing" issue, KVM had a problem dealing with its system resources as well.

When the virtualization host was involved, due either to the load influencing timekeeping directly or making NTP peers temporarily unreachable, the impact on the guests was at its highest level.

4.3.4. THE CRITICALITY OF NTP TIMEKEEPING

The role and importance of the *Network Time Protocol* to guarantee an accurate timekeeping has been progressively "discovered" during the analysis phase of this research.

VMware hints about the capabilities of NTP and recommends it as a good solution for precise timekeeping into Virtual Machines (ref [1] in the bibliography), with a series of insightful hints not only for the ESXi Hypervisor.

Looking at *time* baselines published in Section 8 of the *Results-Attachments* document, the role of NTP is very important even on *idle* systems. Not only the VMware guests show time drift without NTP, but the physical host of OpenVZ does it too. Both, remarkably, show in less than 48 hours a comparable time drift, with one of the VMware guests drifting more than the other (a recurrent situation, might have to do with the CPU core assigned to a VM).

A manual of the Linux OpenSuSE distribution referred to KVM, in one of the scattered sources of information about *kvm-clock* (ref [41]) and in contradiction with RedHat sources (ref [42]), suggests :

When using kvm-clock, it is not recommended to use NTP in the VM Guest, as well.

The results of that approach can unfortunately be very disappointing, as another baseline in *Results-Attachment* shows: the time drifts away very quickly. If NTP is used instead, the different instances in host and guests seem to engage in a fight about who knows the best time: an understandable albeit very unwanted behavior, in fact: *kvm-clock* is supposed to provide controlled access to the time structures of the host but then prevent the conflicts, something that has not happened during this research.

In at least three occasions published among the results, during Network tests in OpenVZ and Combined tests in VMware, a correlation between a "good" server (the time monitoring node) and misleading "bad" servers (the active NTP servers made hard to reach by network load) clearly shows that NTP could be the cause of time drift. Faced with information from the "bad servers" *ntpd* seems to be causing a wrong correction that affects even a physical system like the OpenVZ host (paragraph 6.9 of *Results-Attachments*).

A VMware guest whose network was kept busy (and maybe disrupted by memory load) during combined tests, followed the same fate as OpenVZ, with "bad" NTP servers causing drift followed by relatively fast recovery when "good" query answers were elaborated.

The conclusions drawn by this research that the results published try to demonstrate is that:

when accurate time is critical, a stable source of clock synchronization like NTP is indispensable. NTP could then become the weakest link in accurate timekeeping.

Well-reachable and stable NTP servers have to be chosen, a path to them should be kept available for as much as possible (for example by using separate network interfaces or implementing packet shaping) and a good configuration for the NTP daemon is very important (choosing whether to use "*packet bursts*", testing different "*polling intervals*", etc). Otherwise another clocksource different than remote NTP servers should be used.

4.4. RESOURCE SEPARATION

The following paragraphs contain conclusions about the *resource separation* requirement. Extended results are published in the *Results-Attachments* document where the *Comparative Graphs* probably provide the best figures of resource separation in a graphical form, while *Full Results Tables* can be consulted for further verification.

However the following results attempt to *summarize the results*, knowing more details about the extent of the tests, the kind of results available and the main use cases employed may be necessary for their interpretation. Such information is present in the Introduction of the *Results-Attachments* document and with a general overview in Section 3 of this document, paragraph 3.6.

4.4.1. VMWARE VSPHERE HYPERVISOR (ESXi)

CPU

To the extent of the test executed, the **CPU** performance reported by different types of benchmarks *appeared stable and well balanced among ESXi Virtual Machines*.

In case of simultaneous benchmarking of two guests, both reported similar CPU performance, close to their baseline performance.

In case of competing usage of the CPU by using Logical Cores during simultaneous benchmarks: each guest show a better performance than the baseline power of the *physical* cores, with one guest gaining light advantage (5%). Possibly the *Turbo Boost* technology provided the additional power. Paragraph 4.8.1 has more details, the comparative graphs providing such information are published in Paragraph 2.4 of the *Results-Attachment* document.

DISK

Simultaneous **DISK** benchmarks on VMware ESXi always show little imbalance between guests, with one guest taking longer (around 25%) to complete. The slower guest has the worse performance.

During simultaneous benchmarks the slower guest takes almost twice than during the standalone benchmark to complete. By concurrent activity, the shared disk I/O resource feels the load and loses performance in slight favor of one guest; however, both guests retain more than 80% than their performance scores than in standalone tests, indicating *acceptable resource separation*.

VMware generally offers a stable split of resources, but the overall disk I/O performance of a guest at any time seems relatively poor compared to that of a physical host not using virtualization. This is more visible during the more extended benchmarks of Stage 1.

MEMORY

A VMware guest, in standalone **MEMORY** benchmarks, shows excellent performance, around 80% of the native performance of a physical host. In a simultaneous test the performance drops at about an *equally split* 75% of the baseline on both guests, showing good *resource separation*. The simultaneous test duration is about 40% longer than the standalone test, on both guests.

Memory *comparative graphs* and *test duration* are published in Section 5 of *Results-Attachments*.

NETWORK

The complete abstraction of the network stack of the VMware ESXi hypervisor (with virtual switches and a number of other facilities) produced an *excellent separation between network resources of different VMs*.

With the exception mentioned in paragraph 4.4.4: the impact of network load on one guest is not directly visible on the other in terms of *round trip time* or packet loss for example.

Of course the maximum bandwidth of a physical network interface cannot be exceeded, but the hypervisor, in a way remembering *packet shaping*, seems to always make sure to leave some bandwidth available to its guests and keep them separated and reachable. *Advanced Packet Shaping* with reservation of specific network resources, is also possible (ref [20] in bibliography).

4.4.2. OPENVZ

CPU

The **CPU** performance of an OpenVZ container appeared higher during a standalone benchmark. During some simultaneous benchmarks, one of the containers had slight advantage on CPU performance (less than 5%), but in general *CPU resources were well balanced among the containers*.

In case of competing CPU usage by using Logical Cores, during simultaneous benchmarks, each guest retained the baseline power of the *physical* cores, lower than during the standalone test but balanced among guests.

When the *virtualization host* was loaded in combination with a guest, the guest kept almost the same power like when standing alone, with a 5-10% loss in some cases. In case of competing CPU usage with the guest being assigned physical and Logical Cores, it *gained* advantage on the host. The host of OpenVZ appeared to properly control its CPU usage to guarantee the power to the guest.

DISK

During simultaneous disk benchmarks, OpenVZ guests show only *little unbalanced performance* at both stages of tests. Remarkable is that the performance between the standalone benchmarks and the simultaneous benchmarks falls of about 35% in the worst case.

Combined disk performance does not seem to be the strongest point of OpenVZ and that may be due to the shared file system between guests and host. But the duration of simultaneous disk

benchmarks on both guest is practically equal, at around 30% more than the standalone test, indicating some reasonable resource separation.

When simultaneous disk benchmarks are run on one guest and the virtualization host, their general duration becomes extreme: about 90% more than the standalone test duration (the longest benchmark for a guest lasted 7h20m, versus a standalone test duration of 3h17m).

Concerning the performance, the virtualization host only has limited impact on a guest. While the host performance remains understandably similar to that of a physical host, its guest remains at a value similar to when it was competing with another guest.

An OpenVZ container in the standalone benchmark seems to reach 60/80% of the native disk performance of its host, but not 100%. Comparative graphs for disk benchmarks are published in Section 4.4 of the *Results-Attachments* document.

MEMORY

Standalone **MEMORY** benchmarks on one OpenVZ container report at least in a case *much lower performance* than a physical host.

During simultaneous tests, both the containers tested show an equally low performance, so: *resource separation is good*, albeit at a cost in terms of performance.

When virtualization host and one container are benchmarked, the guest maintains a performance similar to the simultaneous test with another guest.

NETWORK

In OpenVZ there is *no network resource separation* observed. All containers, whose network interfaces are bound to a physical interface on the host, feel the impact of network load equally: when one interface is flooded (like as a consequence of a DDoS attack), that causes longer *round trip times* reaching *all* containers.

The lack of network resource separation also includes the virtualization host: network load on the host is equally felt by the containers, however in OpenVZ the containers always tend to remain minimally reachable and little or no packet loss is observed, even if their host is flooded with traffic.

A fine-tuned packet shaping solution using available tools for Linux might help prevent interfaces to be fully flooded by controlling outbound flows and assigning fixed bandwidth to specific container interfaces or to specific services.

Using a separate network interface for specific containers or binding only the "vulnerable" virtualization host to a separate interface are also good options, useful for example for the host to reach the NTP servers independently from the network traffic on the guests.

4.4.3. KVM

CPU

Considering the results of CPU benchmarks only, *the performance* of two guests during simultaneous benchmarks *appears balanced*, also looking at test duration.

When the virtualization host is involved running simultaneous CPU benchmarks with a guest, *there is negative imbalance*: the host causes the guest to take 30-35% more to complete the benchmarks, with lower performance on some CPU intensive benchmarks. An initial indication that the use of the KVM host for CPU intensive operations may not be the recommended option.

DISK

In the standalone **DISK** benchmarks, KVM guests during stage 1 testing show a surprisingly low performance compared to their own host: less than 30%, despite using the *paravirtualized virtio* driver. *The resource separation appears good* during simultaneous benchmarks, though.

When simultaneous tests are run on the virtualization host and one of the guests, the performance of the latter drops even more dramatically, to less than 20% than that of a physical system. By empirical observation the KVM guest was unresponsive when the disk benchmarks were running on its host.

Concerning test durations: during simultaneous testing of the guests, the second guest, albeit the benchmarks reporting similar scores, takes one additional hour to complete the test.

MEMORY

The **MEMORY** performance reported by a standalone benchmark on a guest of KVM seems the same as that of the physical host. When two guests are benchmarked simultaneously, the performance drops to 75-70% , respectively: *an acceptable split between guests*.

When the host is tested with one of its guests, KVM shows its best: the memory performance of the *host* drops to about 50%, while the guest keeps the same performance as in the other simultaneous test. Duration of all simultaneous tests is a little less than double the standalone test.

NETWORK

Also in KVM all network interfaces of the virtual machine are bound to one or more physical interface on the host. The effect of network load on any guest or on the host is felt practically at the same way by all virtual machines and by the host. KVM has *no separation of network resources*.

In KVM, the host deserves more protection or an entirely separate network interface because of its very powerful and uncontrolled network stack: *intensive traffic to the virtualization host can make all guests unreachable*. This is an important remark when considering whether to use the virtualization host for production services, a choice that is not recommended.

Section 6 of the Results-Attachments documents reports the results of network tests.

Also in this case a well tuned *Packet Shaping* configuration could improve the situation and prevent risks: such a solution is recommended when implementing KVM.

4.4.4. THE INTERFERENCE BETWEEN MEMORY AND NETWORK IN HYPERVISORS

Largely documented and repeatable test results have shown an unexpected phenomenon of resource interference between *memory* and *the network stack*.

On both hypervisor solutions (KVM and VMware), any high load applied on the memory caused an added *delay on the entire network stack of the hypervisor*, apparently affecting *all interfaces of all Virtual Machines*, certainly in VMware.

In the ESXi hypervisor, that was worst affected and more broadly tested for this issue, the memory load is confirmed during combined tests to:

- cause a slightly increased delay between 1 and 2 milliseconds to the network stack
- cause a clear and remarkable *drop in inbound network bandwidth generated by iperf*.

This issue also had a clear impact on the NTP activity for timekeeping during memory benchmarks, at least in VMware. The Results-Attachments document shows in Section 5 and 7 a number of cases and relative monitoring graphs.

The network bandwidth drop that was registered might be due to the implementation of the software packet generator *iperf* rather than to the Hypervisor. This aspect will need to be further checked.

It also has to be said that the memory load causing this issue is artificially very high due to the tests that are executed: much higher than on an average production system.

Depending on the application, and assuming the memory bus is seldom loaded to the extremes, 1 or 2 milliseconds delay on the network may not be a big issue. But when *Internet Measurements* are involved, for example in an *Atlas Software probe*, one millisecond or a less reliable inbound bandwidth to the probe can make a difference. It is not up to this research but to developers and management involved with the RIPE ATLAS project to speculate further about the acceptable impact.

This research has not found the cause of this issue. An extra investigation on cause and full extent of this resource interference and on methods to prevent it are deemed necessary.

Since the commercial VMware vSphere Hypervisor is a product actively used at RIPE NCC and if this issue is present there, too, VMware support could be contacted for clarifications.

From a technical perspective, different configurations of the ESXi 5.1 Hypervisor could be tested, like using a different network driver than *VMXNET3 for the Virtual Machines*. Other tools than *iperf* could be employed to check if the inbound bandwidth disruption is real or limited to *iperf*.

4.5. SCALABLE MANAGEABILITY

Scalability and manageability of virtualization candidates could be observed during the setup of the POC lab and thanks to the experience and the information acquired before and during this project. The following paragraphs will attempt to summarize that information.

4.5.1. VMWARE vSPHERE HYPERVISOR

The freely available ESXi Hypervisor by VMware scores **very good** in terms of *manageability and scalability*, despite only offering a subset of the possibilities of other commercial products from VMware.

Commercial additions to the standalone ESXi are beyond the scope of this project, but they are on the market and can be a safe haven in case of need, for example: product support can be acquired on top of a free ESXi license.

The ESXi hypervisor could be installed on remote *RIPE NCC box* nodes through a semi-automated process involving a shared disk resource, an ISO image and a configuration file (ref [16] in the bibliography). The Hypervisor could be firewalled (ref [20]) and once installed it could be managed through the vCLI tools from a Linux command line (ref [22]).

Use of the graphical management interface for any special configuration, if necessary, could be reduced to the minimum with the experience, and performed from remote from a Windows Virtual Machine in the meantime.

Standard Virtual Machines prepared earlier, for example with a preinstalled Atlas *software probe* on them, could be imported using the *ovftool* (ref [14]). Once up and running with the virtual machines deployed, the Hypervisor might practically never need to be accessed anymore unless to upgrade it to a new version.

In one word: the ESXi hypervisor *works* resiliently, without much intervention or particular maintenance needed.

There is no "usable" virtualization host to maintain and configure in ESXi, however the Hypervisor itself can be limitedly reached through ssh. Support for using standard Linux facility like cfengine or even ssh keys is not present, but often no modification is necessary on the Hypervisor itself. An SNMP daemon can be started on it for limited monitoring.

ESXi products are already in use at RIPE NCC: one last positive aspect in terms of experience among internal engineers, existing contact with VMware, future integration possibilities and so on.

For the RIPE NCC box, probably the most relevant limitation of the freely available "vSphere Hypervisor" could lie in its license (treated in paragraph 4.8.3).

4.5.2. OPENVZ

The manageability score of OpenVZ is overall **good**. However, the total cost of ownership is quite higher than that of VMware: a Linux system must be installed, configured and maintained as well as rules for automatic management with tools like *cfengine*.

OpenVZ needs a pre-existing Linux system and then an external kernel on top of it but it is offered as an *all-in-one* solution. For an open source product it has a consistent central resource for documentation, support and upgrades (ref [30]).

When using a supported Linux distribution, like CentOS, once the OpenVZ kernel and tools are added to the system then setting up the first containers is straightforward (about 10-15 command lines to be up and running). No extensive tweaking of the Linux host system is needed. Most if not all deployment operations can be automated.

During this research, the OpenVZ setup required little or no intervention after the initial configuration, and could easily be ported to another host by just copying all relevant files.

The use of (customized) container templates makes deployment of a new service very easy and that can also be automated through *cfengine*, having care to deploy both system template, container configuration and network configuration at the same time. System engineering activities on the containers may be easier thanks to shared process resources and shared disk space with the host.

The virtualization host needs to be protected from unwanted access, eventually by a firewall setup that could use iptables. Packet shaping may also need to be implemented on the host, together with other desired utilities, adding up to the total cost of ownership.

The biggest limitation in the scalability of OpenVZ is that containers may only run Linux distributions using the *vzkernel*, precluding the deployment of *non-Linux* services in the future.

4.5.3. KVM

The scalability and manageability score of KVM is **mediocre** and has also influenced the choice to *not* recommend this technology. Like mentioned in Section 3, the modular nature of KVM with resources, development and support scattered in different places and with different owners make it a quite wasteful solution, however RedHat Inc. centralizes much information and maintains many tools.

In case of a technical issue or a bug, finding specific support may be tricky. Also the risk exists, like for many opensource products, that any of the components of KVM is less/worse maintained than others. Upgrades seem bound to the respective maintainers.

KVM requires a pre-installed Linux host and an amount of configuration on the virtualization host similar to OpenVZ, but KVM is fully integrated in many standard Linux distributions like CentOS, facilitating the deployment. Differently than with OpenVZ, KVM also requires the disk resources for the guests to be prepared. A number of extra parameters of the system may need to be checked and tuned to the necessities of the KVM guests.

KVM may run any kind of system in a VM. It supports importing OVF images but also many more or less scalable or otherwise "raw" methods to import virtual machines, for example the use of LVM snapshots. This of course has technical advantages and may have scalability disadvantages.

Use of *libvirt* makes deployment of new virtual machines and setup of the platform easier with commonly available command line and graphical management tools. Ad-hoc setting and tweaking of specific configurations may be needed from time to time to overcome bugs or limitations.

KVM and its guests are fully customizable, like the Linux system on which they run. It is up to the engineer implementing it to choose every technical configuration detail, from the processor to emulate to the I/O driver to use. This can be at the same time a big advantage and a disadvantage especially if slightly different, ad-hoc solutions have to be tested and are implemented in remote nodes. And have to be documented and maintained for years to come.

4.5.4. THE MIXED BLESSINGS OF A VIRTUALIZATION HOST

The OpenVZ and KVM technologies both run on top of a full-fledged Linux server, the "virtualization host". The virtualization host needs to be installed, configured, managed, protected and it is of course essential that the host stays stable so that its virtualized guests remain stable.

In case of a RIPE NCC box using any of the two mentioned technologies the virtualization host needs to be deployed and maintained remotely, adding complexity. Many operations related to remote maintenance can be automatically executed by tools like the one used at RIPE NCC, *cfengine*.

The virtualization host is expected to have performance and timekeeping of a physical server and does not suffer from the possible limitations of a virtualized guest. Some of those features have been tested by this research. One of the extra questions asked during this research was whether the virtualization host could be used for production services, like to host a RIPE ATLAS software probe or as Atlas measurement target.

System benchmarks but especially network tests results have shown clear limitations and risks connected to the use of the host, the most remarkable of which are listed below. This research recommends *to not use the virtualization host for production services* and to protect it through firewall rules instead, for access only by RIPE NCC. Other suggestions detailed in paragraph 4.7 could be to implement network *packet shaping* on the host to govern traffic bursts, or to bind the virtualized guests on a *separate network interface* than the one used by the host.

- In its standard setup **OpenVZ** containers share the process table and the filesystem between hosts and guests. A faulty "*killall*" command executed in the host or a faulty or

malicious file deletion or modification may damage the containers, if the host is compromised. Access to the host should then be strictly controlled if using OpenVZ. Experimental versions of OpenVZ support an optimized *loop* file system (called *ploop*) that allows to run the containers from inside an image file, a very limited protection from having all files directly exposed if the image can be encrypted.

- The virtualization host, in the tested implementation of **KVM**, works in many ways as a standalone physical system. CPU and especially Disk and Memory load on the host may severely compromise the performance of the guests that run on it as *user processes* under the *qemu* virtualizer. The network stack of the host does not seem to limit itself in any way: if just the IP address of the host is flooded with traffic, all guests may become unreachable or register packet loss, as network tests have shown. Limiting system and network load on the KVM host to the strict necessary is recommended.

4.6. ADDITIONAL REQUIREMENTS

4.6.1. "SCALABLE UPDATES" (DIFFERENT KERNELS ON EACH GUEST)

Full virtualization solutions (VMware ESXi and KVM) offer the possibility to install any operating system in a guest, allowing entirely separate upgrade patterns. OpenVZ is bound to its container model and unfortunately fails this requirement.

According to its documentation, OpenVZ can run another *Linux Distribution* on a container (like *Ubuntu* or *Debian*) based on a pre-built *container template*. A container appears as a standalone system, can be used and upgraded like a physical system, *but it is limited to run under the vkernel*.

4.6.2. COMPLIANCE WITH EXISTING RIPE NCC SERVICES

The simple requirements for running existing RIPE NCC services (see paragraph 2.5.2) are met by all virtualization candidates.

All guests can run CentOS Linux with one network interface named "*eth0*" with most functionalities of a physical interface (*tcpdump captures, iptables, IPv6*). Unlike KVM and VMware, OpenVZ guests need to be configured ad-hoc with "*veth*" interfaces disabling the default point-to-point *veneth* interface.

Further no other requirements for the *RIPE NCC box* to support existing RIPE NCC services have been provided to this research.

4.7. OUTCOME AND FINAL ADVICE

*Among the virtualization candidates tested, the **VMware vSphere Hypervisor 5.1 (ESXi)** has demonstrated the highest and most consistent level of stability and compliance to all requirements of this research.*

System resources appeared very tightly managed and shared across the Virtual Machines, with a possibly consequent lower performance that should not impact the needs of *RIPE NCC box* services.

Network resources also appeared separated and well managed in the vast majority of cases, with a VM remaining reachable even when the other was subject to the highest possible network load.

Regarding manageability, the *ESXi Hypervisor* is without doubts the best solution for possibly care-free and easy deployment and maintenance of several remote nodes with the lowest possible total cost of ownership.

The standardized solution from VMware allows implementing the *RIPE NCC box* with a minimal amount of customized and *ad-hoc* engineering solutions, something that has made the management of certain legacy services (like *TTM*) time and resource consuming and bound to few single engineers that had the knowledge on them.

For an Hypervisor, timekeeping in ESXi was accurate and stable in a vast number of load cases, however *it cannot be considered strictly compliant with the 1ms requirement at all times.*

The other biggest drawback of the ESXi was *the interference between high memory load and the network stack.*

OpenVZ makes the second best choice, offering acceptable performance in many fields and an acceptable degree of compliance to the requirements except for *network resource separation*, a limitation that can be overcome by separating different services, binding their containers to different network interfaces.

A bit of a disappointment, the performance of system resources in an OpenVZ container did not appear to be near-native as it was expected to be.

This research will not be able to provide a direct advice for the implementation of either of the virtualization technologies tested for the RIPE NCC box, as neither of them can be guaranteed 100% compliant to all requirements.

It is a conclusion of this research that a virtualized guest can host an Atlas Anchor node and other services on the RIPE NCC box, but that is subject to certain conditions and potential compromises to evaluate.

Some initiatives could be taken in the *short term* to fully understand and react to the remaining aspects that condition the choice of a virtualization technology versus the other. *Longer term action* related to the very design of the RIPE NCC box should be considered before a broad deployment.

4.7.1. SHORT TERM ACTION

The first short term action this research would advice is to have the *Legal Department* at RIPE NCC to check the effective legal usability of the ESXI hypervisor at no extra costs on "RIPE NCC box" nodes. Paragraph 4.8.2 below has more details.

Furthermore, the *resource interference between memory and the network stack* shown in *Section 5*, and confirmed in *Section 7* of the *Results-Attachments* document and described in paragraph 4.4.4 in this document, may form a serious technical impediment for the use of the *VMware ESXi hypervisor*.

On the short term, partially quoting paragraph 4.4.4, additional research is necessary to:

- Further investigate and clarify the extent of that issue.
- Make clear if, beyond the tiny network delay, an *inbound bandwidth drop* is a consistent consequence of the issue or is only related to *iperf*.
- If the issue cannot be solved by a fix or different settings on the Hypervisor, consider designing and implementing technical mechanisms to prevent or counteract the root cause (the memory overload).

If *the memory issue, licensing limitations* or other reasons remain strong enough to exclude *VMware* as potential candidate: *OpenVZ* virtualization can be used in the RIPE NCC box instead, with the recommended safeguard of using a separate network interface for the virtualization host, carrying management and NTP traffic.

OpenVZ can be the best short term solution to put the RIPE NCC project quickly on wheels

The existing nodes that are currently been deployed for the *Atlas Anchors pilot project* with *CentOS Linux* can be very easily reconfigured (also automatically, using *cfengine*) to run the *vzkernel* of *OpenVZ* and a number of container templates installed on them.

4.7.2. LONG TERM PERSPECTIVE

Another finding of this research, and in particular of *Network* and *Combined tests*, is the criticality of the NTP mechanism for time synchronization described in paragraph 4.3.4.

During the research, the "*timekeeping stability*" requirement was further specified to admit a *maximum drift of 1ms*. No virtualized guest *neither a physical system based on a standard clock oscillator can guarantee such accuracy* in the long term without an extra synchronization mechanism. The *Test Traffic Measurements* project has taught that lesson several years ago; some *Time Baselines* in Section 8 in *Results-Attachments* confirmed it.

In the currently known design, the *RIPE NCC box* would use NTP peers for time synchronization. Once again the *TTM* project had pinpointed some of the limitations of NTP. This research tried to underline the importance of network resource control and a stable network stack for the NTP mechanism to be effective.

The nodes of the *TTM* project, a precise GPS time source during this project, are going to be further decommissioned in 2013. The measurements of RIPE ATLAS are admittedly bound to a lower precision than *TTM* measurements could achieve since *Atlas Probes* themselves do not have a stable clock.

The next questions could be: *what is the level of compromise in time accuracy that the Science Division considers acceptable for the RIPE NCC box?*

For the long term perspective, the advice this research would like to convey is to *reconsider design and strategies for the RIPE NCC box* project.

- If occasional drift beyond the 1ms limit can be tolerated, either the VMware ESXi (once the memory issue is solved) or OpenVZ can be used, with a number of stratum 1 NTP servers carefully configured in each Virtual Machine or in the virtualization host, respectively.
- If the RIPE NCC box and the *Atlas Anchor node* hosted inside it *cannot* tolerate more than 1ms drift at any time, a different technical solution has to be envisaged.
 - Using a separate network interface on the RIPE NCC box for management and *time synchronization* remains a good compromise.
 - If a second interface cannot be used, relying on careful *packet shaping* might be a customized alternative to make sure the total bandwidth used is kept under control and NTP traffic retains some priority, albeit this could be an even more *expensive solution in terms of scalability* and could not be entirely successful.

If none of the above is acceptable, depending on budget and targets of the project another time synchronization solution should be thought of for the RIPE NCC box. A number of products present on the market offer some options: from tiny GPS antennas able to receive a signal with minimal coverage and easier to hook up to a virtualization host than past solutions (in this case, *not* using the ESXi Hypervisor), to using ultra-precise time synchronization appliances.

4.8. ADDITIONAL REMARKS AND SIDE NOTES

This research has tried to give answer to one main question but has in fact given a wider perspective of the virtualization technologies that were tested, by running a large amount of tests. Many of the results produced are shown or explained between this document and the *Results-Attachments* document, however unfortunately not all fields could be tested and not all results could be properly reported because of limited time and space. A few relevant remarks and side notes will conclude this section.

4.8.1. CPU HYPER-THREADING AND TURBO BOOST FEATURES

Hyper-Threading and Turbo Boost are performance features of modern Intel CPUs. Hyper-Threading allows the split of physical cores into two *Logical Cores*, with the processor performing the scheduling. With *Turbo Boost* the processor can temporarily and dynamically increase its nominal run frequency running faster (like an *overclocked* processor) when the system needs more power.

VMware recommendations suggest to leave the *Turbo and Hyper-Threading* features enabled as it is by default. The guests on the ESXi hypervisor have shown similar performance in Unixbench tests with or without Hyper-Threading, with a clear advantage between 10 and 30% when also logical cores are allocated to virtual machines (page 20 in *Results-Attachments*). It is advised to allocate both physical as well as logical cores for the VMs. The performance of ESXi guests does not improve dramatically when the processor *Turbo Boost* is enabled, remaining similar as when it was disabled.

CPU and Hyper-Threading can be left enabled with OpenVZ virtualization. The performance of both OpenVZ containers increases up to 25% when both Turbo Boost is enabled and Hyper-Threading Logical cores are used, that is the recommended choice. By only enabling Turbo Boost the guests can perform 15% faster. Also in OpenVZ both containers in the POC lab gained clear advantage by allocating Hyper-Threading Logical Cores to them, as if they were physical cores. In any cases with no overcommitting both containers maintained a good CPU resource subdivision.

No different Hyper-Threading scenarios have been tested in KVM. Informal information suggested to disable the *Turbo Boost* feature (and that has been done in the POC lab) to prevent KVM to face unexpected changes in the clock rate of the processor at any time.

4.8.2. ABOUT THE vSPHERE HYPERVISOR LICENSE

The VMware vSphere Hypervisor 5.1 (ESXi) can be freely downloaded from the website of VMware and a maximum number of 999 licenses can be obtained for use in physical servers with up to 32Gb of Ram Memory.

One article of the EULA of the product governs the "*Hosting Rights and Restrictions*" and may clarify whether the product can effectively be used without any additional fees for "*RIPE NCC boxes*" hosted at remote locations, administered by GII and hosting services of *the Science Division*, or not.

A relevant excerpt of the mentioned section of the EULA in effect at the moment of writing is quoted below "as is" *strictly for informative purposes*. It may be subject to change and does not constitute any definitive information from VMware, Inc. if the vSphere Hypervisor 5.1 is chosen for the RIPE NCC box, the legal department of RIPE NCC may want to check the entire licensing terms and eventually seek contact with VMware.

2.2 Hosting Rights and Restrictions. Notwithstanding anything to the contrary in this EULA, You may use the Software to deliver "internally developed applications" as a service to third parties via an internal or external network. An "internally developed application" is: (i) a computer application that You have created or developed, and (ii) a third party computer application(s) that (a) is ancillary to your application-based service, and (b) cannot be accessed directly by end users of your application-based service. [..]

Excerpt VMWARE END USER LICENSE AGREEMENT.

4.8.3. ABOUT THE HARDWARE SPECIFICATION

This research received the hardware specifications for an R320 rack mountable server from Dell that corresponds to the model proposed for the *Atlas Anchor Nodes* pilot project. The full specification is published in the *Results-Attachments document*. Two servers of the same type have been used and tested in the POC Lab and the same systems could be certainly used for the RIPE NCC box.

What is found during the tests is that the memory performance of the model proposed is worse than the earlier R310 model used for RIS boxes: the memory chosen has the best speed (1600Mhz) while the processor has a lower bus frequency (1333).

The current specification proposal counts one CPU with 6 cores. Using a higher number of cores (for example: 8) for the RIPE NCC box hardware could increase the virtualized guests that can run on it without resource overcommit. 8 cores CPUs for the R320 model use a faster 1600Mhz bus speed but that comes at an higher cost between 800 and 900€ retail price at the moment of writing, that may not make the upgrade a good choice.

4.9. OPEN QUESTIONS

This paragraphs presents a list of untreated subjects, open questions and unresolved issues related to this research:

- **Network delay caused by memory load.** The biggest open question concerns the correlation between heavy memory load and network delay in full hypervisor solutions, and in particular in the VMware ESXi hypervisor, treated in paragraph 4.4.4 and 4.7.
- **kvm-clock.** Extensive attempts have been made, many are unaccounted in this report, to pinpoint the cause of the disappointing instability of the *kvm-clock paravirtualized clocksource* of KVM. A bug to the CentOS/RedHat kernel development could be filed about that. It remains an open question whether the instability was related in some ways to a kernel bug or to an implementation issue.
- **CPU overcommitment.** Based on the preparatory phase, an assumption has been made about a limited number of services expected to run on a RIPE NCC box, possibly no more than three or four in some years. The minimum requirement for the services often referred to two CPU cores. At this embryonic phase the RIPE NCC boxes with the current hardware specification (same as for *Atlas Anchor Nodes*) were assumed to not need over commitment of CPU resources (i.e. allocating to virtualization guests more CPU cores than physical cores available). CPU overcommitment has not been treated by this research.
- **CPU pinning.** A feature that is instead researched and implemented but only during initial trials (especially when testing with *kvm-clock*) is *CPU pinning*, or assigning specific CPU cores to specific VMs. After feedback from GII engineers about the scalability of pinning and seen the limited number of services expected to run initially on a RIPE NCC box, the use of this feature was not considered critical to this research. That became especially true when CPU cores automatic allocation and resource separation appeared well supported by the virtualization candidates.

Both VMware and KVM seamlessly support pinning of CPU physical or logical cores. OpenVZ does not support pinning but supports setting hard limits on the CPU resources assigned to the guests by using the "cpulimit" option. A number of unaccounted tests during this research have shown the "cpulimit" option to be very effective.

- **Packet Shaping (or Linux Traffic Control)** on Linux Virtualization hosts using the tool *tc* can be implemented to improve resource separation or form a certain protection against network overflow, useful for example for those candidates with no network resource separation like OpenVZ. The VMware ESXi hypervisor also offers packet shaping facilities. No packet shaping solutions have been tested by this research.
- **IPv6.** basic support of all virtualization candidates for IPv6 is checked, all hosts and guests in the POC lab had IPv6 addresses assigned, further this research did not perform any test using IPv6.

- **Jumbo Frames.** Support of the virtualization candidates for Jumbo frames (i.e. Ethernet frames bigger than the standard MTU of 1500 bytes and up to 9000 bytes big) has not been checked. The request about this feature came from potential hosting partners but the support for it at this stage on the RIPE NCC box was considered premature.
- **VLAN tagging.** There was one proposal to check and eventually test the support of the virtualization candidates for 802.1q VLAN tagging, in case multiple guests of a RIPE NCC box sharing the same network interface would need to reside on different VLANs. CentOS Linux (the host of OpenVZ and KVM) supports VLAN tagging natively. The VMware ESXi also fully support VLAN tagging for its VMs. This feature was not tested as also considered premature at this stage.

4.10. VIRTUALIZATION PROTOTYPE AND FUTURE OF THE POC LAB

Two working prototypes of the RIPE NCC box on Dell R320 hardware have been left installed with respectively the VMware vSphere Hypervisor 5.1 and OpenVZ virtualization in the POC lab, with two guests running on them. They remained available to GII together with monitoring host and time monitoring node for further testing and access to test results.

Full support is offered to the manager GII for knowledge transfer to system engineers about the continued use of the POC lab and of the entire testing and monitoring infrastructure should further trials and tests for the RIPE NCC box need to be performed in the near future.

5. BIBLIOGRAPHY

Next to articles, papers and documents consulted as reference, the bibliography also contains links to single relevant reference web pages consulted, like Knowledge Base articles or Documentation pages of products or software used in this research. The latter two are indicated by the prefix [Ref] and are mentioned as an helpful reference for the implementation of the solutions used in this document.

This project has made extensive use of all sorts of Web resources, also as implementation reference. As common by web resources, not all information sources can be fully accounted or considered authoritative for the information that they provide. All documents, trademarks, company names belong to their respective authors or owners.

URLs specified in this Bibliography are openly listed by search engines and freely reachable as of January 2013. They are reported 'as-is' without guarantees of any kind and without any implied responsibilities about the status or reason of their publishing on the web at the given addresses.

NTP and Timekeeping

- [1] *Timekeepng in VMware Virtual Machines, VMware INC*
<http://www.vmware.com/files/pdf/Timekeeping-In-VirtualMachines.pdf>
- [2] *RFC 5905 - Network Time Protocol Version 4 -*
<http://www.ietf.org/rfc/rfc5905.txt>
- [3] *David L. Mills, Internet Time SynchronizationL The Network Time Protocol, 1991, IEEE*
<http://www.cs.sunysb.edu/~jgao/CSE590-spring11/91-ntp.pdf>
- [4] *Clock Jitter and Measurement, SITime, 6 February 2009*
<http://www.sitime.com/support2/documents/AN10007-Jitter-and-measurement.pdf>
- [5] *Various Authors, PC Based Precision Timing Without GPS*
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.86.4804&rep=rep1&type=pdf>
- [6] *Various Authors, Precision Synchronization of Computer Network Clocks*
<http://www.eecis.udel.edu/~mills/database/papers/fine.pdf>
- [7] *Svein Johannessen, Time Synchronization in a Local Area Network*
http://www-lar.deis.unibo.it/people/crossi/files/SCD/Time_synchronization_in_a_local_area_network.pdf
- [8] *Various Authors, Integration of high accurate Clock Synchronization into Ethernet-based Distributed Systems*
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.19.5588&rep=rep1&type=pdf>
- [9] *[Ref] NTP Documentation Archive -* <http://doc.ntp.org/>
- [10] *[Ref] Monitoring and Controlling NTP*
<http://support.ntp.org/bin/view/Support/MonitoringAndControllingNTP>
- [11] *[Ref] NTP Timestamp Calculations* <http://www.eecis.udel.edu/~mills/time.html>

- [12] [Ref] <http://support.ntp.org/bin/view/Servers/StratumOneTimeServers>

VMware ESXi

- [13] *What's New in VMware vSphere 5.1 Performance, VMware Inc.*
<http://www.vmware.com/files/pdf/techpaper/Whats-New-VMware-vSphere-5.1-Performance-Technical-Whitepaper.pdf>
- [14] *Performance Best Practice for VMware vSphere 5.0, VMware Inc.*
http://www.vmware.com/pdf/Perf_Best_Practices_vSphere5.0.pdf
- [15] *OVF Tool User Guide, VMware*
http://www.vmware.com/support/developer/ovf/ovf10/ovftool_10_userguide.pdf
- [16] [Ref] *KB 2004582 - Deploying ESXi 5.x using the Scripted Install feature, VMware Inc.*
http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=2004582
- [17] [Ref] *KB 1006427 - Timekeeping best practices for Linux Guests, VMware Inc.*
http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1006427
- [18] [Ref] *KB 1005092 - Troubleshooting NTP on ESX and ESXi, VMware Inc.*
http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1005092
- [19] [Ref] *KB 1017910 - Using Tech Support Mode in ESXi 4.1 and ESXi 5.x*
http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1017910
- [20] [Ref] *KB 2005284 - About the ESXi 5.x firewall*
http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=2005284
- [21] [Ref] *Traffic Shaping with VMware ESX Server*
http://www.petri.co.il/traffic_shaping_with_vmware_esx_server.htm
- [22] [Ref] *vSphere Command-Line Interface Documentation*
<http://www.vmware.com/support/developer/vcli/>
- [23] [Ref] *vSphere Management Assistant*
<http://www.vmware.com/support/developer/vima/>
- [24] [Ref] *vSphere 5.1 Documentation*
<http://pubs.vmware.com/vsphere-51/index.jsp>
- [25] [Ref] *Memory management techniques in VMware vSphere*
<http://expertpandas.com/blog/index.php/memory-management-techniques-in-vmware-vsphere/>
- [26] [Ref] *How to update ESXi server to ESXi 5.1*
<http://www.v-front.de/2012/09/how-to-update-your-free-whitebox-esxi.html>

OpenVZ

- [27] *Kirill Kolyshkin, Virtualization in Linux, OpenVZ*
<http://download.openvz.org/doc/openvz-intro.pdf>
- [28] *Various Authors, Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors -*
http://nsg.cs.princeton.edu/publication/vserver_eurosys_07.pdf
- [29] [Ref] *CentOS wiki OpenVZ*
<http://wiki.centos.org/HowTos/Virtualization/OpenVZ>
- [30] [Ref] *Differences between venet and veth, OpenVZ*
http://wiki.openvz.org/Differences_between_venet_and_veth

[31] [Ref] *Traffic Shaping with tc*
http://wiki.openvz.org/Traffic_shaping_with_tc

[32] [Ref] <http://wiki.openvz.org>

KVM

[33] *Virtualization Host Configuration and Guest Installation Guide, Red Hat, Inc, 2012*
https://access.redhat.com/knowledge/docs/en-US/Red_Hat_Enterprise_Linux/6/html/Virtualization_Host_Configuration_and_Guest_Installation_Guide/

[34] *Kernel Virtual Machine (KVM) - Best practices for KVM, IBM*
http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/topic/liaat/liaatbestpractices_pdf.pdf

[35] *Quick Start Guide for installing and running KVM, IBM*
http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/topic/liaai/kvminstall/kvminstall_pdf.pdf

[36] *KVM Virtualization in RHEL 6 Made Easy, Dell*
http://linux.dell.com/files/whitepapers/KVM_Virtualization_in_RHEL_6_made_easy.pdf

[37] [Ref] *Installing a KVM Guest OS from the Command-Line (virt-install), Tectopia*
[http://www.techotopia.com/index.php/Installing_a_KVM_Guest_OS_from_the_Command-Line_\(virt-install\)](http://www.techotopia.com/index.php/Installing_a_KVM_Guest_OS_from_the_Command-Line_(virt-install))

[38] [Ref] <http://www.linux-kvm.org/>

[39] [Ref] <http://wiki.centos.org/HowTos/Virtualization/Introduction>

[40] [Ref] *KVM is Linux. Xen is Not.*
<http://chucknology.com/2012/02/02/kvm-is-linux-xen-is-not/>

[41] [Ref] *Managing Clock in KVM, opensuse.org*
<http://doc.opensuse.org/documentation/html/openSUSE/opensuse-kvm/cha.libvirt.config.html#sec.kvm.managing.clock>

[42] *KVM Virtual Machine Timing Management, RedHat Enterprise Virtualization for Servers*
https://access.redhat.com/knowledge/docs/en-US/Red_Hat_Enterprise_Virtualization_for_Servers/2.2/html/Administration_Guide/chap-Virtualization-KVM_guest_timing_management.html

Gnuplot and Monitoring

[43] [Ref] *Smokeping* <http://oss.oetiker.ch/smokeping/>

[44] [Ref] *Gnuplot* - <http://www.gnuplot.info>

[45] [Ref] <http://www.observium.org>

[46] [Ref] <http://www.cacti.org>

Testing and Benchmarking

[47] [Ref] <http://www.phoronix-test-suite.com/>

[48] [Ref] <http://openbenchmarking.org>

[49] [Ref] <http://iperf.fr>

[50] [Ref] <http://code.google.com/p/byte-unixbench/>

[51] [Ref] <http://weather.ou.edu/~apw/projects/stress/>

Others - Hardware - OS

[52] *Kempen, Keizer - "Competent afstuderen en stage lopen", Norshoff Uitgevers 2009*

- [53] *Using vFlash on iDRAC 6, Dell*
http://media.community.dell.com/en/dtc/attach/using_vflash_on_idrac6_final.docx
- [54] [Ref] <http://www.intel.com/content/www/us/en/architecture-and-technology/turbo-boost/turbo-boost-technology.html>
- [55] [Ref] <http://www.intel.com/content/www/us/en/architecture-and-technology/hyper-threading/hyper-threading-technology.html>
- [56] [Ref] *Dell Remote Access Controller, DELL*
<http://en.community.dell.com/techcenter/systems-management/w/wiki/3204.dell-remote-access-controller-drac-idrac.aspx>
- [57] [Ref] *Everything you need to know about the CPU C-States Power Saving Modes*
<http://www.hardwaresecrets.com/article/611>
- [58] [Ref] *Libvirt* - <http://libvirt.org>
- [59] [Ref] *CentOS wiki* - <http://wiki.centos.org>

